Oracle® Text

Reference

11*g* Release 2 (11.2)

E24436-02

August 2011

ORACLE®

Oracle Text Reference, 11*g* Release 2 (11.2)

E24436-02

Primary Author: Cathy Shea

Contributor: Mohammad Faisal, Roger Ford, Paul Lane, Wesley Lin, Yasuhiro Matsuda

Contents

3 Oracle Text CONTAINS Query Operators

8 CTX_DOC Package

12 CTX_THES Package

13 CTX_ULEXER Package

14 Oracle Text Utilities

B Oracle Text Supported Document Formats

C Text Loading Examples for Oracle Text

D Oracle Text Multilingual Features

E Oracle Text Supplied Stoplists

F The Oracle Text Scoring Algorithm

G Oracle Text Views

H Stopword Transformations in Oracle Text

Index

Preface

This Preface contains these topics:

Audience

Oracle Text Reference is intended for application developers or system administrators who maintain an Oracle Text system in an Oracle environment. To use this document, you need experience with Oracle Database, SQL, SQL*Plus, and PL/SQL.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information about Oracle Text, see:

- *Oracle Text Application Developer's Guide*

For more information about Oracle Database, see:

- *Oracle Database Concepts*
- *Oracle Database Administrator's Guide*
- *Oracle Database Utilities*
- *Oracle Database Performance Tuning Guide*

- *Oracle Database SQL Language Reference*
- *Oracle Database Reference*
- *Oracle Database Advanced Application Developer's Guide*

For more information about PL/SQL, see:

- *Oracle Database PL/SQL Language Reference*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in Oracle Text?

The following describes new features of the Oracle Database 11*g* Release 2 (11.2) edition of Oracle Text and provides pointers to additional information. Information about new features from previous releases is also retained to help you migrate to the current release.

The following sections describe the new features in Oracle Text:

Oracle Database 11*g* Release 2 (11.2) New Features in Oracle Text

- Name matching

  Someone accustomed to the spelling rules of one culture can have difficulty applying those same rules to a name originating from a different culture. Name matching provides a solution to match proper names that might differ in spelling due to orthographic variation. It also enables you to search for somewhat inaccurate data, such as might occur when a record's first name and surname are not propertly segmented.

  See Also: *Oracle Text Application Developer's Guide* for further information

- Result set interface

  A page of search results typically consists of many disparate elements, such as metadata of the first few documents, per-word hit counts, or total hit counts. In past releases of Oracle Text, generating these results required several queries and calls, such as a query on the base table, a call to `CTX_QUERY.COUNT_HITS`, and so on. Each call required time to reparse the query and look up index metadata. In this release, instead of accessing the database to construct bits of the search results, you can use the result set interface, which is able to produce the various kinds of data needed for a page of search results all at once, thus improving performance by sharing overhead. The result set interface can also return data views that are difficult to express in SQL, such as top N by category queries.

  See Also: *Oracle Text Application Developer's Guide* for further information

Oracle Database 11*g* Release 1 (11.1) New Features in Oracle Text

- On Windows systems, the executable file that you specify for the `USER_FILTER command` attribute must now exist in the `%ORACLE_HOME%/ctx/bin` directory instead of `%ORACLE_HOME%/bin`.

  See Also: USER_FILTER on page 2-24

- Zero downtime for applications with new incremental indexing and online index creation.

See Also: "Creating a CONTEXT Index Incrementally with POPULATE_PENDING" in *Oracle Text Application Developer's Guide*

- New features for re-creating an index online and finer control for maintenance processes.

  See Also: "Re-Creating an Index" in *Oracle Text Application Developer's Guide*

- New Oracle Text Manager in Oracle Enterprise Manager with which you can:
  - Monitor health of Oracle Text indexes.
  - Modify index settings.
  - Generate index-level statistics about disk space, fragmentation, garbage, frequency of words, and more.
  - Synchronize, optimize, and rebuild indexes.
  - Diagnose problems, and resume failed operations.
  - Manage logs.

  See Also: "Text Manager in Oracle Enterprise Manager" in *Oracle Text Application Developer's Guide*

- New support for composite domain index for `CONTEXT` indextype for improved mixed-query performance.

  See Also: "Composite Domain Index (CDI) in Oracle Text" in *Oracle Text Application Developer's Guide*

- Improved query performance and scalability.

  See Also: "Parallelizing Queries Across Oracle RAC Nodes" in *Oracle Text Application Developer's Guide*

- New `SDATA` section type and `SDATA` operator that enable range searches on metadata.

  See Also: "SDATA Section" in *Oracle Text Application Developer's Guide*

- New user-defined scoring feature, `DEFINESCORE` and `DEFINEMERGE`.

  See Also: "DEFINESCORE" on page 3-13 and "DEFINEMERGE" on page 3-12

- New values for the `INDEX_STEMS` attribute of the `BASIC_LEXER` type to enable better query performance for stem ($) queries.

  See Also: "BASIC_LEXER" on page 2-31

- `NOPOPULATE` option for `ALTER INDEX` to support incremental indexing.

  See Also: "POPULATE | NOPOPULATE" on page 1-46

- New limit for the number of partitions in Oracle Text is now the same as the maximum for Oracle Database.

  See Also: "Partitioned Tables and Indexes" in *Oracle Text Application Developer's Guide*

- New usage tracking feature.

  See Also: "Database Feature Usage Tracking in Oracle Enterprise Manager" in *Oracle Text Application Developer's Guide*

1

Oracle Text SQL Statements and Operators

This chapter describes the SQL statements and Oracle Text operators for creating and managing Oracle Text indexes and performing Oracle Text queries.

The following statements are described in this chapter:

ALTER INDEX

> **Note:** This section describes the ALTER INDEX statement as it pertains to managing an Oracle Text domain index.
>
> For a complete description of the ALTER INDEX statement, see *Oracle Database SQL Language Reference*.

Purpose

Use ALTER INDEX to make changes to, or perform maintenance tasks for a CONTEXT, CTXCAT, or CTXRULE index.

All Index Types

Use ALTER INDEX to perform the following tasks on all Oracle Text index types:

- Rename the index or index partition. See "ALTER INDEX RENAME Syntax" on page 1-4.
- Rebuild the index using different preferences. Some restrictions apply for the CTXCAT index type. See "ALTER INDEX REBUILD Syntax" on page 1-4.
- Add stopwords to the index. See "ALTER INDEX REBUILD Syntax" on page 1-4.

CONTEXT and CTXRULE Index Types

Use ALTER INDEX to perform the following tasks on CONTEXT and CTXRULE index types:

- Resume a failed index operation (creation/optimization).
- Add sections and stop sections to the index.
- Replace index metadata.

> **See Also:** "ALTER INDEX REBUILD Syntax" on page 1-4 to learn more about performing these tasks

Overview of ALTER INDEX Syntax

The syntax for ALTER INDEX is fairly complex. The major divisions are covered in the following sections:

- "ALTER INDEX MODIFY PARTITION Syntax" on page 1-3—use this to modify an index partition's metadata.
- "ALTER INDEX PARAMETERS Syntax" on page 1-3—use this to modify the parameters of a nonpartitioned index, or to modify all partitions of a local partitioned index, without rebuilding the index.
- "ALTER INDEX RENAME Syntax" on page 1-4—use this to rename an index or index partition.
- "ALTER INDEX REBUILD Syntax" on page 1-4—use this to rebuild an index or index partition. With this statement, you can also replace index metadata; add stopwords, sections, and stop sections to an index; and resume a failed operation.

  The parameters for ALTER INDEX REBUILD have their own syntax, which is a subset of the syntax for ALTER INDEX. For example, the ALTER INDEX REBUILD

PARAMETERS statement can take either REPLACE or RESUME as an argument, and ALTER INDEX REBUILD PARAMETERS ('REPLACE') can take several arguments. Valid examples of ALTER INDEX REBUILD include the following statements:

```
ALTER INDEX REBUILD PARALLEL n
ALTER INDEX REBUILD PARAMETERS ('SYNC memsize')
ALTER INDEX REBUILD PARAMETERS ('REPLACE DATASTORE datastore_pref')
ALTER INDEX REBUILD PARAMETERS ('REPLACE WORDLIST wordlist_pref')
```

ALTER INDEX MODIFY PARTITION Syntax

Use the following syntax to modify the metadata of an index partition:

```
ALTER INDEX index_name MODIFY PARTITION partition_name PARAMETER (paramstring)
```

index_name
Specify the name of the index whose partition metadata you want to modify.

partition_name
Specify the name of the index partition whose metadata you want to modify.

paramstring
The only valid argument here is 'REPLACE METADATA'. This follows the same syntax as ALTER INDEX REBUILD PARTITION PARAMETERS ('REPLACE METADATA'); see the REPLACE METADATA subsection of the "ALTER INDEX REBUILD Syntax" section on page 1-6 for more information. (The two statements are equivalent. ALTER INDEX MODIFY PARTITION is offered for ease of use, and is the recommended syntax.)

ALTER INDEX PARAMETERS Syntax

Use the following syntax to modify the parameters either of nonpartitioned or local partitioned indexes, without rebuilding the index. For partitioned indexes, this statement works at the index level, not at the partition level. This statement changes information for the entire index, including all partitions.

```
ALTER INDEX index_name PARAMETERS (paramstring)
```

paramstring
ALTER INDEX PARAMETERS accepts the following arguments for *paramstring*:

- 'REPLACE METADATA'

  Replaces current metadata. See the REPLACE METADATA subsection of the "ALTER INDEX REBUILD Syntax" section on page 1-6 for more information.

- 'ADD STOPWORD'

  Dynamically adds a stopword to an index. See the ADD STOPWORD subsection of the "ALTER INDEX REBUILD Syntax" section on page 1-10 for more information.

- 'ADD FIELD SECTION'

  Dynamically adds a field section to an index. See the ADD FIELD subsection of the "ALTER INDEX REBUILD Syntax" section on page 1-10 for more information.

- 'ADD ZONE SECTION'

  Dynamically adds a zone section to an index. See the ADD ZONE subsection of the "ALTER INDEX REBUILD Syntax" section on page 1-10 for more information.

- 'ADD ATTR SECTION'

Dynamically adds an attribute section to an index. See the ADD ATTR subsection of the "ALTER INDEX REBUILD Syntax" section on page 1-11 for more information.

Each of the prior statements has an equivalent ALTER INDEX REBUILD PARAMETERS version. For example, ALTER INDEX PARAMETERS ('REPLACE METADATA') is equivalent to ALTER INDEX REBUILD PARAMETERS ('REPLACE METADATA'). However, the ALTER INDEX PARAMETERS versions work on either partitioned or nonpartitioned indexes, whereas the ALTER INDEX REBUILD PARAMETERS versions work only on nonpartitioned indexes.

ALTER INDEX RENAME Syntax

Use the following syntax to rename an index or index partition:

```
ALTER INDEX [schema.]index_name RENAME TO new_index_name;

ALTER INDEX [schema.]index_name RENAME PARTITION part_name TO new_part_name;
```

[schema.]index_name
Specify the name of the index to rename.

new_index_name
Specify the new name for schema.index. The new_index_name parameter can be no more than 25 bytes, and 21 bytes for a partitioned index. If you specify a name longer than 25 bytes (or longer than 21 bytes for a partitioned index), then Oracle Text returns an error and the renamed index is no longer valid.

Note: When new_index_name is more than 25 bytes (21 for local partitioned index) and less than 30 bytes, Oracle Text renames the index, even though the system returns an error. To drop the index and associated tables, you must drop new_index_name with the DROP INDEX statement and then re-create and drop index_name.

---

part_name
Specify the name of the index partition to rename.

new_part_name
Specify the new name for partition.

ALTER INDEX REBUILD Syntax

Use ALTER INDEX REBUILD to rebuild an index, rebuild an index partition, resume a failed operation, replace index metadata, add stopwords to an index, or add sections and stop sections to an index.

The ALTER INDEX REBUILD syntax has its own subsyntax. That is, its parameters have their own syntax. For example, the ALTER INDEX REBUILD PARAMETERS statement can take either REPLACE or RESUME as an argument, and ALTER INDEX REBUILD PARAMETERS ('REPLACE') has several arguments it can take.

Valid examples of ALTER INDEX REBUILD include the following statements:

```
ALTER INDEX REBUILD PARALLEL n
ALTER INDEX REBUILD PARAMETERS (SYNC memsize)
ALTER INDEX REBUILD PARAMETERS (REPLACE DATASTORE datastore_pref)
ALTER INDEX REBUILD PARAMETERS (REPLACE WORDLIST wordlist_pref)
```

This is the syntax for ALTER INDEX REBUILD:

```
ALTER INDEX [schema.]index [REBUILD] [PARTITION partname] [ONLINE] [PARAMETERS
(paramstring)][PARALLEL N];
```

PARTITION *partname*

Rebuilds the index partition partname. Only one index partition can be built at a time.

When you rebuild a partition you can specify only RESUME or REPLACE in paramstring. These operations work only on the partname you specify.

With the REPLACE operation, you can specify only MEMORY and STORAGE for each index partition.

Adding Partitions To add a partition to the base table, use the ALTER TABLE SQL statement. When you add a partition to an indexed table, Oracle Text automatically creates the metadata for the new index partition. The new index partition has the same name as the new table partition. Change the index partition name with ALTER INDEX RENAME.

Splitting or Merging Partitions Splitting or merging a table partition with ALTER TABLE renders the index partitions invalid. You must rebuild them with ALTER INDEX REBUILD.

[ONLINE]

Enables you to continue to perform updates, insertions, and deletions on a base table. It does not enable you to query the base table.

> **Note:** You can specify REPLACE or RESUME when rebuilding an index or an index partition ONLINE.

PARAMETERS (paramstring)

Optionally specify paramstring. If you do not specify paramstring, then Oracle Text rebuilds the index with existing preference settings.

The syntax for paramstring is as follows:

```
paramstring =
'REPLACE
     [DATASTORE datastore_pref]
     [FILTER filter_pref]
     [LEXER lexer_pref]
     [WORDLIST wordlist_pref]
     [STORAGE storage_pref]
     [STOPLIST stoplist]
     [SECTION GROUP section_group]
     [MEMORY memsize
     [[POPULATE | NOPOPULATE]
     [INDEX SET index_set]

     [METADATA preference new_preference]
     [METADATA FORMAT COLUMN format_column_name]
     [[METADATA] SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)]
     [[METADATA] TRANSACTIONAL|NONTRANSACTIONAL

| RESUME [memory memsize]
| OPTIMIZE [token index_token | fast | full [maxtime (time | unlimited)]
| SYNC [memory memsize]
| ADD STOPWORD word [language language]
```

```
| ADD ZONE SECTION section_name tag tag
| ADD FIELD SECTION section_name tag tag [(VISIBLE | INVISIBLE)]
| ADD ATTR SECTION section_name tag tag@attr
| ADD STOP SECTION tag'
```

REPLACE [*optional_preference_list*]
Rebuilds an index. You can optionally specify your own preferences, or system-defined preferences.

You can replace only preferences that are supported for that index type. For instance, you cannot replace index set for a `CONTEXT` or `CTXRULE` index. Similarly, for the `CTXCAT` index type, you can replace lexer, wordlist, storage index set, and memory preferences.

The `POPULATE` parameter is the default and need not be specified. If you want to empty the index of its contents, then specify `NOPOPULATE`. Clear an index of its contents when you must rebuild your index incrementally. The `NOPOPULATE` choice is available for a specific partition of the index, and not just for the entire index.

If you are rebuilding a partitioned index using the `REPLACE` parameter, then you can specify only `STORAGE`, `MEMORY`, and `NOPOPULATE`.

> **See Also:** Chapter 2, "Oracle Text Indexing Elements" for more information about creating and setting preferences, including information about system-defined preferences

REPLACE METADATA *preference new_preference*
Replaces the existing `preference` class settings, including `SYNC` parameters, of the index with the settings from `new_preference`. Only index preferences and attributes are replaced. The index is not rebuilt.

This statement is useful for when you want to replace a preference and its attribute settings after the index is built, without reindexing all data. Reindexing data can require significant time and computing resources.

This statement is also useful for changing the `SYNC` parameter type, which can be automatic, manual, or on-commit.

The `ALTER INDEX REBUILD PARAMETER ('REPLACE METADATA')` statement does not work for a local partitioned index at the global level for the index. You cannot, for example, use this syntax to change a global preference, such as filter or lexer type, without rebuilding the index. Use `ALTER INDEX PARAMETERS` instead to change the metadata of an index at the global level, including all partitions. See "ALTER INDEX PARAMETERS Syntax" on page 1-3.

When should I use the METADATA keyword? `REPLACE METADATA` should be used only when the change in index metadata will not lead to an inconsistent index, which can lead to incorrect query results.

For example, use this statement in the following instances:

- To go from a single-language lexer to a multilexer in anticipation of multilingual data. For an example, see "Replacing Index Metadata: Changing Single-Lexer to Multilexer" on page 1-13.
- To change the `WILDCARD_MAXTERMS` setting in BASIC_WORDLIST.
- To change the `SYNC` parameter type, which can be automatic, manual, or on-commit.

These changes are safe and will not lead to an inconsistent index that might adversely affect your query results.

Caution: The REPLACE METADATA statement can result in inconsistent index data, which can lead to incorrect query results. As such, Oracle does not recommend using this statement, unless you carefully consider the effect it will have on the consistency of your index data and subsequent queries.

There can be many instances when changing metadata can result in inconsistent index data. For example, Oracle recommends *against* using the METADATA keyword after performing the following procedures:

- Changing the USER_DATASTORE procedure to a new PL/SQL stored procedure that has different output.
- Changing the BASIC_WORDLIST attribute PREFIX_INDEX from NO to YES because no prefixes have been generated for existing documents. Changing it from YES to NO is safe.
- Adding or changing BASIC_LEXER printjoin and skipjoin characters, because new queries with these characters would be lexed differently from how these characters were lexed at index time.

In these unsafe cases, Oracle recommends rebuilding the index.

REPLACE [METADATA] SYNC (MANUAL | EVERY "*interval-string*" | ON COMMIT)
Specifies SYNC for automatic synchronization of the CONTEXT index when a DML change has occurred to the base table. You can specify one of the SYNC methods shown in Table 1–1.

Table 1–1 ALTER INDEX SYNC Methods

SYNC Type	Description
MANUAL	Means no automatic synchronization. This is the default. You must manually synchronize the index using CTX_DDL.SYNC_INDEX. Use MANUAL to disable ON COMMIT and EVERY synchronization.
EVERY *interval-string*	Automatically synchronize the index at a regular interval specified by the value of *interval-string*, which takes the same syntax as that for scheduler jobs. Automatic synchronization using EVERY requires that the index creator have CREATE JOB privileges. Ensure that *interval-string* is set to a long enough period so that any previous synchronization jobs will have completed. Otherwise, the synchronization job may hang. The *interval-string* argument must be enclosed in double quotation marks (" "). See "Enabling Automatic Index Synchronization" on page 1-48 for an example of automatic synchronization syntax.

Table 1–1 (Cont.) ALTER INDEX SYNC Methods

SYNC Type	Description
ON COMMIT	Synchronize the index immediately after a commit. The commit does not return until the sync is complete. (Because the synchronization is performed as a separate transaction, there may be a time period, usually small, when the data is committed but index changes are not.) The operation uses the memory specified with the *memory* parameter. Note that the sync operation has its own transaction context. If this operation fails, the data transaction still commits. Index synchronization errors are logged in the CTX_USER_INDEX_ERRORS view. See "Viewing Index Errors" under CREATE INDEX. See "Enabling Automatic Index Synchronization" on page 1-48 for an example of ON COMMIT syntax.

Each partition of a locally partitioned index can have its own type of sync: (ON COMMIT, EVERY, or MANUAL). The type of sync specified in master parameter strings applies to all index partitions unless a partition specifies its own type.

With automatic (EVERY) synchronization, you can specify memory size and parallel synchronization. The syntax is:

```
... EVERY interval_string MEMORY mem_size PARALLEL paradegree ...
```

ON COMMIT synchronizations can only be executed serially and at the same memory size as what was specified at index creation.

Note: This command rebuilds the index. When you want to change the SYNC setting without rebuilding the index, use the REBUILD REPLACE METADATA SYNC (MANUAL | ON COMMIT) operation.

REPLACE [METADATA] TRANSACTIONAL | NONTRANSACTIONAL

This parameter enables you to turn the TRANSACTIONAL property on or off. For more information, see "TRANSACTIONAL" on page 1-47.

Using this parameter only succeeds if there are no rows in the DML pending queue. Therefore, you may need to sync the index before issuing this command.

To turn on the TRANSACTIONAL index property:

```
ALTER INDEX myidx REBUILD PARAMETERS('replace metadata transactional');
```

or

```
ALTER INDEX myidx REBUILD PARAMETERS('replace  transactional');
```

To turn off the TRANSACTIONAL index property:

```
ALTER INDEX myidx REBUILD PARAMETERS('replace metadata nontransactional');
```

or

```
ALTER INDEX myidx REBUILD PARAMETERS('replace  nontransactional');
```

RESUME [MEMORY *memsize*]
Resumes a failed index operation. You can optionally specify the amount of memory to use with `memsize`.

> **Note:** This `ALTER INDEX` operation applies only to `CONTEXT` and `CTXRULE` indexes. It does not apply to `CTXCAT` indexes.

OPTIMIZE [token index_token | fast | full [maxtime (time | unlimited)]

> **Note:** This `ALTER INDEX` operation will not be supported in future releases.
>
> To optimize your index, use CTX_DDL.OPTIMIZE_INDEX.

Optimizes the index. Specify `token`, `fast`, or `full` optimization. You typically optimize after you synchronize the index.

When you optimize in `token` mode, Oracle Text optimizes only `index_token`. Use this method of optimization to quickly optimize index information for specific words.

When you optimize in `fast` mode, Oracle Text works on the entire index, compacting fragmented rows. However, in `fast` mode, old data is not removed.

When you optimize in `full` mode, you can optimize the whole index or a portion. This method compacts rows and removes old data (deleted rows).

> **Note:** Optimizing in `full` mode runs even when there are no deleted document rows. This is useful when you must optimize time-limited batches with the `maxtime` parameter.

Use the `maxtime` parameter to specify in minutes the time Oracle Text is to spend on the optimization operation. Oracle Text starts the optimization where it left off and optimizes until complete or until the time limit has been reached, whichever comes first. Specifying a time limit is useful for automating index optimization, where you set Oracle Text to optimize the index for a specified time on a regular basis.

When you specify `maxtime unlimited`, the entire index is optimized. This is the default. When you specify 0 for `maxtime`, Oracle Text performs minimal optimization.

Log the progress of optimization by writing periodic progress updates to the `CTX_OUTPUT` log. An event for `CTX_OUTPUT.ADD_EVENT`, called `CTX_OUTPUT.EVENT_OPT_PRINT_TOKEN`, prints each token as it is being optimized.

> **Note:** This `ALTER INDEX` operation applies only to `CONTEXT` and `CTXRULE` indexes. It does not apply to `CTXCAT` indexes.

SYNC [MEMORY *memsize*]

> **Note:** This `ALTER INDEX` operation will not be supported in future releases.
>
> To synchronize your index, use CTX_DDL.SYNC_INDEX.

Synchronizes the index. You can optionally specify the amount of run-time memory to use with `memsize`. Synchronize the index when you have DML operations on your base table.

> **Note:** This `ALTER INDEX` operation applies only to `CONTEXT` and `CTXRULE` indexes. It does not apply to `CTXCAT` indexes.

Memory Considerations The memory parameter memsize specifies the amount of memory Oracle Text uses for the `ALTER INDEX` operation before flushing the index to disk. Specifying a large amount of memory improves indexing performance because there is less I/O and improves query performance and maintenance because there is less fragmentation.

Specifying smaller amounts of memory increases disk I/O and index fragmentation, but might be useful if you want to track indexing progress or when run-time memory is scarce.

ADD STOPWORD *word* [language *language*]
Dynamically adds a stopword word to the index.

Index entries for `word` that existed before this operation are not deleted. However, subsequent queries on `word` are treated as though it has always been a stopword.

When your stoplist is a multilanguage stoplist, you must specify `language`.

The index is *not* rebuilt by this statement.

ADD ZONE SECTION *section_name* tag *tag*
Dynamically adds the zone section `section_name` identified by `tag` to the existing index.

The added section `section_name` applies only to documents indexed after this operation. For the change to take effect, you must manually re-index any existing documents that contain the tag.

The index is *not* rebuilt by this statement.

> **Note:** This `ALTER INDEX` operation applies only to `CONTEXT` and `CTXRULE` indexes. It does not apply to `CTXCAT` indexes.

> **See Also:** "Notes" on page 1-15

ADD FIELD SECTION *section_name* tag *tag* [(VISIBLE | INVISIBLE)]
Dynamically adds the field section `section_name` identified by `tag` to the existing index.

Optionally specify `VISIBLE` to make the field sections visible. The default is `INVISIBLE`.

> **See Also:** CTX_DDL.ADD_FIELD_SECTION for more information on visible and invisible field sections.

The added section `section_name` applies only to documents indexed after this operation. For the change to affect previously indexed documents, you must explicitly re-index the documents that contain the tag.

This statement does not rebuild the index.

Note: This ALTER INDEX operation applies only to CONTEXT CTXRULE indexes. It does not apply to CTXCAT indexes.

See Also: "Notes" on page 1-15

ADD ATTR SECTION *section_name* tag *tag@attr*

Dynamically adds an attribute section section_name to the existing index. You must specify the XML tag and attribute in the form tag@attr. You can add attribute sections only to XML section groups.

The added attribute section section_name applies only to documents indexed after this operation. For the change to take effect, you must manually re-index any existing documents that contain the tag.

The index is *not* rebuilt by this statement.

Note: This ALTER INDEX operation applies only to CONTEXT CTXRULE indexes. It does not apply to CTXCAT indexes.

See Also: "Notes" on page 1-15.

ADD STOP SECTION *tag*

Dynamically adds the stop section identified by tag to the existing index. As stop sections apply only to automatic sectioning of XML documents, the index must use the AUTO_SECTION_GROUP section group. The tag you specify must be case sensitive and unique within the automatic section group or else ALTER INDEX raises an error.

The added stop section tag applies only to documents indexed after this operation. For the change to affect previously indexed documents, you must explicitly re-index the documents that contain the tag.

The text within a stop section can always searched.

The number of stop sections you can add is unlimited.

The index is *not* rebuilt by this statement.

See Also: "Notes" on page 1-15

Note: This ALTER INDEX operation applies only to CONTEXT indexes. It does not apply to CTXCAT indexes.

PARALLEL n

Using *n*, you can optionally specify the parallel degree for parallel indexing. This parameter is supported only when you use SYNC, REPLACE, and RESUME in paramstring. The actual degree of parallelism might be smaller depending on your resources.

Parallel indexing can speed up indexing when you have large amounts of data to index and when your operating system supports multiple CPUs.

ALTER INDEX Examples

Resuming Failed Index

The following statement resumes the indexing operation on `newsindex` with 2 megabytes of memory:

```
ALTER INDEX newsindex REBUILD PARAMETERS('resume memory 2M');
```

Rebuilding an Index

The following statement rebuilds the index, replacing the stoplist preference with `new_stop`.

```
ALTER INDEX newsindex REBUILD PARAMETERS('replace stoplist new_stop');
```

Rebuilding a Partitioned Index

The following example creates a partitioned text table, populates it, and creates a partitioned index. It then adds a new partition to the table and rebuilds the index with `ALTER INDEX` as follows:

```
PROMPT create partitioned table and populate it

create table part_tab (a int, b varchar2(40)) partition by range(a)
(partition p_tab1 values less than (10),
 partition p_tab2 values less than (20),
 partition p_tab3 values less than (30));

insert into part_tab values (1,'Actinidia deliciosa');
insert into part_tab values (8,'Distictis buccinatoria');
insert into part_tab values (12,'Actinidia quinata');
insert into part_tab values (18,'Distictis Rivers');
insert into part_tab values (21,'pandorea jasminoides Lady Di');
insert into part_tab values (28,'pandorea rosea');

commit;

PROMPT create partitioned index
create index part_idx on part_tab(b) indextype is ctxsys.context
local (partition p_idx1, partition p_idx2, partition p_idx3);


PROMPT add a partition and populate it
alter table part_tab add partition p_tab4 values less than (40);
insert into part_tab values (32, 'passiflora citrina');
insert into part_tab values (33, 'passiflora alatocaerulea');
commit;
```

The following statement rebuilds the index in the newly populated partition. In general, the index partition name for a newly added partition is the same as the table partition name, unless the name has already been used. In this case, Oracle Text generates a new name.

```
alter index part_idx rebuild partition p_tab4;
```

The following statement queries the table for the two hits in the newly added partition:

```
select * from part_tab where contains(b,'passiflora') >0;
```

The following statement queries the newly added partition directly:

```
select * from part_tab partition (p_tab4) where contains(b,'passiflora') >;
```

Replacing Index Metadata: Changing Single-Lexer to Multilexer

The following example demonstrates how an application can migrate from single-language documents (English) to multilanguage documents (English and Spanish) by replacing the index metadata for the lexer.

```
REM creates a simple table, which stores only English (American) text

create table simple (text varchar2(80));
insert into simple values ('the quick brown fox');
commit;

REM create a simple lexer to lex this English text

begin
  ctx_ddl.create_preference('us_lexer','basic_lexer');
end;
/

REM create a text index on the simple table
create index simple_idx on simple(text)
indextype is ctxsys.context parameters ('lexer us_lexer');

REM we can query easily
select * from simple where contains(text, 'fox')>0;

REM now suppose we want to start accepting Spanish documents.
REM first we have to extend the table with a language column
alter table simple add (lang varchar2(10) default 'us');

REM now let's create a Spanish lexer,
begin
  ctx_ddl.create_preference('e_lexer','basic_lexer');
  ctx_ddl.set_attribute('e_lexer','base_letter','yes');
end;
/
REM Then create a multilexer incorporating our English and Spanish lexers.
REM Note that the DEFAULT lexer is the exact same lexer, with which we have
REM have already indexed all the documents.
begin
  ctx_ddl.create_preference('m_lexer','multi_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','default','us_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','spanish','e_lexer');
end;
/
REM next replace our metadata
alter index simple_idx rebuild
parameters ('replace metadata language column lang lexer m_lexer');

REM We are ready for some Spanish data.  Note that we could have inserted
REM this BEFORE the alter index, as long as we did not SYNC.
insert into simple values ('el zorro marr&oacute;n r&aacute;pido', 'e');
commit;
exec ctx_ddl.sync_index('simple_idx');
REM now query the Spanish data with base lettering:
select * from simple where contains(text, 'rapido')>0;
```

Optimizing the Index

To optimize your index, use CTX_DDL.OPTIMIZE_INDEX.

Synchronizing the Index

To synchronize your index, use CTX_DDL.SYNC_INDEX.

Adding a Zone Section

To add to the index the zone section `author` identified by the tag `<author>`, enter the following statement:

```
ALTER INDEX myindex REBUILD PARAMETERS('add zone section author tag author');
```

Adding a Stop Section

To add a stop section identified by tag `<fluff>` to the index that uses the `AUTO_SECTION_GROUP`, enter the following statement:

```
ALTER INDEX myindex REBUILD PARAMETERS('add stop section fluff');
```

Adding an Attribute Section

Assume that the following text appears in an XML document:

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

Assume also that you want to create a separate section for the title attribute and you want to name the new attribute section `booktitle`. To do so, enter the following statement:

```
ALTER INDEX myindex REBUILD PARAMETERS('add attr section booktitle tag
title@book');
```

Using Flashback Queries

If a Text query is flashed back to a point before an `ALTER INDEX` statement was issued on the Text index for which the query is being run, then:

- The query optimizer will not choose the index access path for that given index because the index is treated according to its creation time with `ALTER INDEX`. Therefore, to the query optimizer, the index is perceived not to exist.
- The functional processing of the Text operator will fail with ORA-01466 or ORA-08176 errors if the `ALTER INDEX` statement involves re-creation of `DR$` index tables.

To work around this issue, use the `DBMS_FLASHBACK` package. For example:

```
EXEC dbms_flashback.enable_at_system_change_number(:scn);
SELECT id from documents WHERE CONTAINS(text, 'oracle')>0;
EXEC dbms_flashback.disable;
```

Note: In previous releases, flashback Text queries using `AS OF` predicates with Text operators such as `CONTAINS` and `CATSEARCH` are not supported.

See Also: "Using DBMS_FLASHBACK Package" in *Oracle Database Advanced Application Developer's Guide*

Notes

Add Section Constraints

Before altering the index section information, Oracle Text checks the new section against the existing sections to ensure that all validity constraints are met. These constraints are the same for adding a section to a section group with the `CTX_DDL` PL/SQL package and are as follows:

- You cannot add zone, field, or stop sections to a `NULL_SECTION_GROUP`.
- You cannot add zone, field, or attribute sections to an automatic section group.
- You cannot add attribute sections to anything other than XML section groups.
- You cannot have the same tag for two different sections.
- Section names for zone, field, and attribute sections cannot intersect.
- You cannot exceed 64 fields per section.
- You cannot add stop sections to basic, HTML, XML, or news section groups.
- `SENTENCE` and `PARAGRAPH` are reserved section names.

Related Topics

CTX_DDL.SYNC_INDEX in Chapter 7, "CTX_DDL Package"

CTX_DDL.OPTIMIZE_INDEX in Chapter 7, "CTX_DDL Package"

CREATE INDEX on page 1-36

ALTER TABLE: Supported Partitioning Statements

> **Note:** This section describes the ALTER TABLE statement as it pertains to adding and modifying a partitioned text table with a context domain index.
>
> For a complete description of the ALTER TABLE statement, see *Oracle Database SQL Language Reference*.

Purpose

Use the ALTER TABLE statement to add, modify, split, merge, exchange, or drop a partitioned text table with a context domain index. The following sections describe some of the ALTER TABLE operations.

Modify Partition Syntax

Unusable Local Indexes

```
ALTER TABLE  [schema.]table MODIFY PARTITION partition UNUSABLE LOCAL INDEXES
```

Marks the index partition corresponding to the given table partition UNUSABLE. You might mark an index partition unusable before you rebuild the index partition as described in "Rebuild Unusable Local Indexes".

If the index partition is not marked unusable, then the statement returns without actually rebuilding the local index partition.

Rebuild Unusable Local Indexes

```
ALTER TABLE  [schema.]table MODIFY PARTITION partition REBUILD UNUSABLE LOCAL
INDEXES
```

Rebuilds the index partition corresponding to the specified table partition that has an UNUSABLE status.

> **Note:** If the index partition status is already VALID before you enter this statement, then this statement does *not* rebuild the index partition. Do not depend on this statement to rebuild the index partition unless the index partition status is UNUSABLE.

Add Partition Syntax

```
ALTER TABLE [schema.]table ADD PARTITION [partition]
VALUES LESS THAN (value_list) [partition_description]
```

Adds a new partition to the high end of a range-partitioned table.

To add a partition to the beginning or to the middle of the table, use the ALTER TABLE SPLIT PARTITION statement.

The newly added table partition is always empty, and the context domain index (if any) status for this partition is always VALID. After issuing DML, if you want to synchronize or optimize this newly added index partition, then you must look up the index partition name and enter the ALTER INDEX REBUILD PARTITION statement.

For this newly added partition, the index partition name is usually the same as the table partition name, but if the table partition name is already used by another index partition, the system assigns a name in the form of SYS_Pn.

By querying the USER_IND_PARTITIONS view and comparing the HIGH_VALUE field, you can determine the index partition name for the newly added partition.

Merge Partition Syntax

```
ALTER TABLE [schema.]table
MERGE PARTITIONS partition1, partition2
[INTO PARTITION [new_partition] [partition_description]]
[UPDATE GLOBAL INDEXES]
```

Applies only to a range partition. This statement merges the contents of two adjacent partitions into a new partition and then drops the original two partitions. If the resulting partition is non-empty, then the corresponding local domain index partition is marked UNUSABLE. You can use ALTER TABLE MODIFY PARTITION to rebuild the partition index.

For a global, nonpartitioned index, if you perform the merge operation without an UPDATE GLOBAL INDEXES clause, then the resulting index (if not NULL) will be invalid and must be rebuilt. If you specify the UPDATE GLOBAL INDEXES clause after the operation and the SYNC type is MANUAL, then the index will be valid, but you still must synchronize the index with CTX_DDL.SYNC_INDEX for the update to take place.

The naming convention for the resulting index partition is the same as in the ALTER TABLE ADD PARTITION statement.

Split Partition Syntax

```
ALTER TABLE [schema.]table
SPLIT PARTITION partition_name_old
AT (value_list)
[into (partition_description, partition_description)]
[prallel_clause]
[UPDATE GLOBAL INDEXES]
```

Applies only to range partitions. This statement divides a table partition into two partitions, thus adding a new partition to the table. The local corresponding index partitions will be marked UNUSABLE if the corresponding table partitions are non-empty. Use the ALTER TABLE MODIFY PARTITION statement to rebuild the partition indexes.

For a global, nonpartitioned index, if you perform the split operation without an UPDATE GLOBAL INDEXES clause, then the resulting index (if not NULL) will be invalid and must be rebuilt. If you specify the UPDATE GLOBAL INDEXES clause after the operation and the SYNC type is MANUAL, then the index will be valid, but you still must synchronize the index with CTX_DDL.SYNC_INDEX for the update to take place.

The naming convention for the two resulting index partition is the same as in the ALTER TABLE ADD PARTITION statement.

Exchange Partition Syntax

```
ALTER TABLE [schema.]table EXCHANGE PARTITION partition WITH TABLE table
[INCLUDING|EXCLUDING INDEXES}
[WITH|WITHOUT VALIDATION]
```

```
[EXCEPTIONS INTO [schema.]table]
[UPDATE GLOBAL INDEXES]
```

Converts a partition to a nonpartitioned table, and converts a table to a partition of a partitioned table by exchanging their data segments. Rowids are preserved.

If EXCLUDING INDEXES is specified, all the context indexes corresponding to the partition and all the indexes on the exchanged table are marked as UNUSABLE. To rebuild the new index partition in this case, issue an ALTER TABLE MODIFY PARTITION statement.

If INCLUDING INDEXES is specified, then for every local domain index on the partitioned table, there must be a nonpartitioned domain index on the nonpartitioned table. The local index partitions are exchanged with the corresponding regular indexes.

For a global, nonpartitioned index, if you perform the exchange operation without an UPDATE GLOBAL INDEXES clause, then the resulting index (if not NULL) will be invalid and must be rebuilt. If you specify the UPDATE GLOBAL INDEXES clause after the operation and the SYNC type is MANUAL, then the index will be valid, but you still must synchronize the index with CTX_DDL.SYNC_INDEX for the update to take place.

Field Sections

Field section queries might not work the same way if the nonpartitioned index and local index use different section IDs for the same field section.

Storage

Storage is not changed. So if the index on the nonpartitioned table $I table was in tablespace XYZ, then after the exchange partition, it will still be in tablespace XYZ, but now it is the $I table for an index partition.

Storage preferences are not switched, so if you switch and then rebuild the index, then the table may be created in a different location.

Restrictions

Both indexes must be equivalent. They must use the same objects and the same settings for each object. Note that Oracle Text checks only that the indexes are using the same object. But they should use the same exact everything.

No index object can be partitioned, that is, when the user has used the storage object to partition the $I, $N tables.

If either index or index partition does not meet all these restrictions an error is raised and both the index and index partition will be INVALID. You must manually rebuild both index and index partition using the ALTER INDEX REBUILD statement.

Truncate Partition Syntax

```
ALTER TABLE [schema.]table TRUNCATE PARTITION [DROP|REUSE STORAGE] [UPDATE GLOBAL
INDEXES]
```

Removes all rows from a partition in a table. Corresponding CONTEXT index partitions are also removed.

For a global, nonpartitioned index, if you perform the truncate operation without an UPDATE GLOBAL INDEXES clause, then the resulting index (if not NULL) will be invalid and must be rebuilt. If you specify the UPDATE GLOBAL INDEXES clause after the operation, the index will be valid.

ALTER TABLE Examples

Global Index on Partitioned Table Examples

The following example creates a range-partitioned table with three partitions. Each partition is populated with two rows. A global, nonpartitioned CONTEXT index is then created. To demonstrate the UPDATE GLOBAL INDEXES clause, the partitions are split and merged with an index synchronization.

```
create table tdrexglb_part(a int, b varchar2(40)) partition by range(a)
(partition p1 values less than (10),
 partition p2 values less than (20),
 partition p3 values less than (30));

insert into tdrexglb_part values (1,'row1');
insert into tdrexglb_part values (8,'row2');
insert into tdrexglb_part values (11,'row11');
insert into tdrexglb_part values (18,'row18');
insert into tdrexglb_part values (21,'row21');
insert into tdrexglb_part values (28,'row28');

commit;
create index tdrexglb_parti on tdrexglb_part(b) indextype is ctxsys.context;

create table tdrexglb(a int, b varchar2(40));

insert into tdrexglb values(20,'newrow20');
commit;


PROMPT make sure query works
select * from tdrexglb_part where contains(b,'row18') >0;

PROMPT split partition
alter table tdrexglb_part split partition p2 at (15) into
(partition p21, partition p22) update global indexes;

PROMPT before sync
select * from tdrexglb_part where contains(b,'row11') >0;
select * from tdrexglb_part where contains(b,'row18') >0;

exec ctx_ddl.sync_index('tdrexglb_parti')

PROMPT after sync
select * from tdrexglb_part where contains(b,'row11') >0;
select * from tdrexglb_part where contains(b,'row18') >0;

PROMPT merge partition
alter table tdrexglb_part merge partitions p22, p3
into partition pnew3 update global indexes;

PROMPT before sync
select * from tdrexglb_part where contains(b,'row18') >0;
select * from tdrexglb_part where contains(b,'row28') >0;
exec ctx_ddl.sync_index('tdrexglb_parti');

PROMPT after sync
select * from tdrexglb_part where contains(b,'row18') >0;
select * from tdrexglb_part where contains(b,'row28') >0;
```

```
PROMPT drop partition
alter table tdrexglb_part drop partition p1 update global indexes;

PROMPT before sync
select * from tdrexglb_part where contains(b,'row1') >0;
exec ctx_ddl.sync_index('tdrexglb_parti');

PROMPT after sync
select * from tdrexglb_part where contains(b,'row1') >0;

PROMPT exchange partition
alter table tdrexglb_part exchange partition pnew3 with table
tdrexglb update global indexes;

PROMPT before sync
select * from tdrexglb_part where contains(b,'newrow20') >0;
select * from tdrexglb_part where contains(b,'row28') >0;

exec ctx_ddl.sync_index('tdrexglb_parti');
PROMPT after sync
select * from tdrexglb_part where contains(b,'newrow20') >0;
select * from tdrexglb_part where contains(b,'row28') >0;

PROMPT move table partition
alter table tdrexglb_part move partition p21 update global indexes;
PROMPT before sync
select * from tdrexglb_part where contains(b,'row11') >0;

exec ctx_ddl.sync_index('tdrexglb_parti');
PROMPT after sync
select * from tdrexglb_part where contains(b,'row11') >0;

PROMPT truncate table partition
alter table tdrexglb_part truncate partition p21 update global indexes;

update global indexes;
```

CATSEARCH

Use the CATSEARCH operator to search CTXCAT indexes. Use this operator in the WHERE clause of a SELECT statement.

The CATSEARCH operator also supports database links. You can identify a remote table or materialized view by appending @dblink to the end of its name. The dblink must be a complete or partial name for a database link to the database containing the remote table or materialized view. (Indexing of remote views is not supported.)

The grammar of this operator is called CTXCAT. You can also use the CONTEXT grammar if your search criteria require special functionality, such as thesaurus, fuzzy matching, proximity searching, or stemming. To utilize the CONTEXT grammar, use the Query Template Specification in the text_query parameter as described in this section.

About Performance

Use the CATSEARCH operator with a CTXCAT index mainly to improve mixed-query performance. Specify your text query condition with text_query and your structured condition with the structured_query argument.

Internally, Oracle Text uses a combined B-tree index on text and structured columns to quickly produce results satisfying the query.

Limitations

If the optimizer chooses to use the functional query invocation, then your query will fail. The optimizer might choose functional invocation when your structured clause is highly selective.

The structured_query argument of the CATSEARCH operator must reference columns used during CREATE INDEX sets; otherwise, error DRG-10845 will be raised. For example, the error will be raised if you issue a CATSEARCH query on a view created on top of a table with the CTXCAT index on it, and the name of the logical column on the view is different from the actual column name on the physical table. The columns referenced by the structured_query argument of the CATSEARCH operator must be the physical column name used during CREATE INDEX sets, not the logical column on the view.

Syntax

```
CATSEARCH(
[schema.]column,
text_query       [VARCHAR2|CLOB],
structured_query VARCHAR2,
RETURN NUMBER;
```

[schema.]column

Specifies the text column to be searched on. This column must have a CTXCAT index associated with it.

text_query

Specify one of the following to define your search in column:

- CATSEARCH Query Operations
- Query Template Specification (for using CONTEXT grammar)

CATSEARCH Query Operations

The CATSEARCH operator supports only the following query operations:

- Logical AND
- Logical OR (|)
- Logical NOT (-)
- " " (quoted phrases)
- Wildcarding

Table 1–2 provides the syntax for these operators.

Table 1–2 CATSEARCH Query Operators

Operation	Syntax	Description of Operation
Logical AND	a b c	Returns rows that contain a, b, and c.
Logical OR	a | b | c	Returns rows that contain a, b, or c.
Logical NOT	a - b	Returns rows that contain a and not b.
Hyphen with no space	a-b	Hyphen treated as a regular character. For example, if the hyphen is defined as skipjoin, words such as *web-site* are treated as the single query term *website*. Likewise, if the hyphen is defined as a printjoin, words such as *web-site* are treated as *web-site* in the CTXCAT query language.
" "	"a b c"	Returns rows that contain the phrase "a b c". For example, entering "Sony CD Player" means return all rows that contain this sequence of words.
( )	(A B) | C	Parentheses group operations. This query is equivalent to the CONTAINS query (A &B) | C.
Wildcard (right and double truncated)	term* a*b	The wildcard character matches zero or more characters. For example, *do** matches *dog*, and *gl*s* matches *glass*. Left truncation not supported. Note: Oracle recommends that you create a prefix index if your application uses wildcard searching. Set prefix indexing with the BASIC_WORDLIST preference.

The following limitations apply to these operators:

- The left-hand side (the column name) must be a column named in at least one of the indexes of the index set.
- The left-hand side must be a plain column name. Functions and expressions are not allowed.
- The right-hand side must be composed of literal values. Functions, expressions, other columns, and subselects are not allowed.
- Multiple criteria can be combined with AND. Note that OR is not supported.
- When querying a remote table through a database link, the database link must be specified for CATSEARCH as well as for the table being queried.

For example, these expressions are supported:

```
catsearch(text, 'dog', 'foo > 15')
catsearch(text, 'dog', 'bar = ''SMITH''')
catsearch(text, 'dog', 'foo between 1 and 15')
catsearch(text, 'dog', 'foo = 1 and abc = 123')
catsearch@remote(text, 'dog', 'foo = 1 and abc = 123')
```

These expressions are not supported:

```
catsearch(text, 'dog', 'upper(bar) = ''A''')
catsearch(text, 'dog', 'bar LIKE ''A%''')
catsearch(text, 'dog', 'foo = abc')
catsearch(text, 'dog', 'foo = 1 or abc = 3')
```

Query Template Specification

Specifies a marked-up string that specifies a query template. Specify one of the following templates:

- Query rewrite, used to expand a query string into different versions
- Progressive relaxation, used to progressively enter less restrictive versions of a query to increase recall
- Alternate grammar, used to specify `CONTAINS` operators (See "CONTEXT Query Grammar Examples" on page 1-25)
- Alternate language, used to specify alternate query language
- Alternate scoring, used to specify alternate scoring algorithms

> **See Also:** The text_query parameter description for `CONTAINS` on page 1-28 for more information about the syntax for these query templates

structured_query

Specifies the structured conditions and the `ORDER BY` clause. There must exist an index for any column you specify. For example, if you specify `'category_id=1 order by bid_close'`, you must have an index for `'category_id, bid_close'` as specified with the `CTX_DDL.ADD_INDEX` package.

With `structured_query`, you can use standard SQL syntax only with the following operators:

- =
- <=
- >=
- >
- <
- IN
- BETWEEN
- AND (to combine two or more clauses)

> **Note:** You cannot use parentheses () in the `structured_query` parameter.

Examples

1. Create the table.

The following statement creates the table to be indexed:

```
CREATE TABLE auction (category_id number primary key, title varchar2(20),
bid_close date);
```

The following table inserts the values into the table:

```
INSERT INTO auction values(1, 'Sony CD Player', '20-FEB-2000');
INSERT INTO auction values(2, 'Sony CD Player', '24-FEB-2000');
INSERT INTO auction values(3, 'Pioneer DVD Player', '25-FEB-2000');
INSERT INTO auction values(4, 'Sony CD Player', '25-FEB-2000');
INSERT INTO auction values(5, 'Bose Speaker', '22-FEB-2000');
INSERT INTO auction values(6, 'Tascam CD Burner', '25-FEB-2000');
INSERT INTO auction values(7, 'Nikon digital camera', '22-FEB-2000');
INSERT INTO auction values(8, 'Canon digital camera', '26-FEB-2000');
```

1. Create the `CTXCAT` index:

The following statements create the `CTXCAT` index:

```
begin
ctx_ddl.create_index_set('auction_iset');
ctx_ddl.add_index('auction_iset','bid_close');
end;
/
CREATE INDEX auction_titlex ON auction(title) INDEXTYPE IS CTXSYS.CTXCAT
PARAMETERS ('index set auction_iset');
```

1. Query the Table:

A typical query with `CATSEARCH` might include a structured clause as follows to find all rows that contain the word *camera* ordered by `bid_close`:

```
SELECT * FROM auction WHERE CATSEARCH(title, 'camera', 'order by bid_close desc')>
0;

CATEGORY_ID TITLE               BID_CLOSE
----------- ------------------- ---------
          8 Canon digital camera 26-FEB-00
          7 Nikon digital camera 22-FEB-00
```

The following query finds all rows that contain the phrase *Sony CD Player* and that have a bid close date of February 20, 2000:

```
SELECT * FROM auction WHERE CATSEARCH(title, '"Sony CD Player"',
'bid_close=''20-FEB-00''')> 0;

CATEGORY_ID TITLE               BID_CLOSE
----------- ------------------- ---------
          1 Sony CD Player      20-FEB-00
```

The following query finds all rows with the terms *Sony* and *CD* and *Player*:

```
SELECT * FROM auction WHERE CATSEARCH(title, 'Sony CD Player', 'order by bid_close
desc')> 0;
CATEGORY_ID TITLE               BID_CLOSE
----------- ------------------- ---------
          4 Sony CD Player      25-FEB-00
          2 Sony CD Player      24-FEB-00
          1 Sony CD Player      20-FEB-00
```

The following query finds all rows with the term *CD* and not *Player*:

```
SELECT * FROM auction WHERE CATSEARCH(title, 'CD - Player', 'order by bid_close
desc')> 0;

CATEGORY_ID TITLE               BID_CLOSE
----------- ------------------- ---------
          6 Tascam CD Burner    25-FEB-00
```

The following query finds all rows with the terms *CD* or *DVD* or *Speaker*:

```
SELECT * FROM auction WHERE CATSEARCH(title, 'CD | DVD | Speaker', 'order by
bid_close desc')> 0;

CATEGORY_ID TITLE               BID_CLOSE
----------- ------------------- ---------
          3 Pioneer DVD Player  25-FEB-00
          4 Sony CD Player      25-FEB-00
          6 Tascam CD Burner    25-FEB-00
          2 Sony CD Player      24-FEB-00
          5 Bose Speaker        22-FEB-00
          1 Sony CD Player      20-FEB-00
```

The following query finds all rows that are about *audio equipment*:

```
SELECT * FROM auction WHERE CATSEARCH(title, 'ABOUT(audio equipment)', NULL)> 0;
```

CONTEXT Query Grammar Examples

The following examples show how to specify the `CONTEXT` grammar in `CATSEARCH` queries using the template feature:

```
PROMPT
PROMPT fuzzy: query = ?test
PROMPT should match all fuzzy variations of test (for example, text)
select pk||' ==> '||text from test
where catsearch(text,
'<query>
  <textquery grammar="context">
     ?test
  </textquery>
</query>','')>0
order by pk;

PROMPT
PROMPT fuzzy: query = !sail
PROMPT should match all soundex variations of bot (for example, sell)
select pk||' ==> '||text from test
where catsearch(text,
'<query>
  <textquery grammar="context">
     !sail
  </textquery>
</query>','')>0
order by pk;

PROMPT
PROMPT theme (ABOUT) query
PROMPT query: about(California)
select pk||' ==> '||text from test
where catsearch(text,
```

```
'<query>
  <textquery grammar="context">
     about(California)
  </textquery>
</query>','')>0
order by pk;
```

The following example shows a field section search against a CTXCAT index using CONTEXT grammar by means of a query template in a CATSEARCH query:

```
-- Create and populate table
create table BOOKS (ID number, INFO varchar2(200), PUBDATE DATE);

insert into BOOKS values(1, '<author>NOAM CHOMSKY</author><subject>CIVIL
   RIGHTS</subject><language>ENGLISH</language><publisher>MIT
  PRESS</publisher>', '01-NOV-2003');

insert into BOOKS values(2, '<author>NICANOR PARRA</author><subject>POEMS
  AND ANTIPOEMS</subject><language>SPANISH</language>
  <publisher>VASQUEZ</publisher>', '01-JAN-2001');

insert into BOOKS values(1, '<author>LUC SANTE</author><subject>XML
  DATABASE</subject><language>FRENCH</language><publisher>FREE
  PRESS</publisher>', '15-MAY-2002');

commit;

-- Create index set and section group
exec ctx_ddl.create_index_set('BOOK_INDEX_SET');
exec ctx_ddl.add_index('BOOKSET','PUBDATE');

exec ctx_ddl.create_section_group('BOOK_SECTION_GROUP',
      'BASIC_SECTION_GROUP');
exec ctx_ddl.add_field_section('BOOK_SECTION_GROUP','AUTHOR','AUTHOR');
exec ctx_ddl.add_field_section('BOOK_SECTION_GROUP','SUBJECT','SUBJECT');
exec ctx_ddl.add_field_section('BOOK_SECTION_GROUP','LANGUAGE','LANGUAGE');
exec ctx_ddl.add_field_section('BOOK_SECTION_GROUP','PUBLISHER','PUBLISHER');


-- Create index
create index books_index on books(info) indextype is ctxsys.ctxcat
  parameters('index set book_index_set section group book_section_group');

-- Use the index
-- Note that: even though CTXCAT index can be created with field sections, it
-- cannot be accessed using CTXCAT grammar (default for CATSEARCH).
-- We need to use query template with CONTEXT grammar to access field
-- sections with CATSEARCH.

select  id, info from books
where catsearch(info,
'<query>
      <textquery grammar="context">
              NOAM within author and english within language
      </textquery>
 </query>',
'order by pubdate')>0;
```

Related Topics

"Syntax for CTXCAT Index Type" on page 1-52

Oracle Text Application Developer's Guide

CONTAINS

Use the CONTAINS operator in the WHERE clause of a SELECT statement to specify the query expression for a Text query.

The CONTAINS operator also supports database links. You can identify a remote table or materialized view by appending @dblink to the end of its name. The dblink must be a complete or partial name for a database link to the database containing the remote table or materialized view. (Querying of remote views is not supported.)

CONTAINS returns a relevance score for every row selected. Obtain this score with the SCORE operator.

The grammar for this operator is called the CONTEXT grammar. You can also use CTXCAT grammar if your application works better with simpler syntax. To do so, use the Query Template Specification in the text_query parameter as described in this section.

> **See Also:** "The CONTEXT Grammar" topic in *Oracle Text Application Developer's Guide*

Syntax

```
CONTAINS(
         [schema.]column,
         text_query     [VARCHAR2|CLOB]
         [,label        NUMBER])
RETURN NUMBER;
```

[schema.]column
Specify the text column to be searched on. This column must have a Text index associated with it.

text_query
Specify one of the following:

- The query expression that defines your search in column.
- A marked-up document that specifies a query template. Use one of the following templates:

Query Rewrite Template

Use this template to automatically write different versions of a query before you submit the query to Oracle Text. This is useful when you need to maximize the recall of a user query. For example, you can program your application to expand a single phrase query of 'cat dog' into the following queries:

```
{cat} {dog}
{cat} ; {dog}
{cat} AND {dog}
{cat} ACCUM {dog}
```

These queries are submitted as one query and results are returned with no duplication. In this example, the query returns documents that contain the phrase *cat dog* as well as documents in which *cat* is near *dog*, and documents that have *cat* and *dog*.

This is done with the following template:

```
 <query>
   <textquery lang="ENGLISH" grammar="CONTEXT"> cat dog
     <progression>
       <seq><rewrite>transform((TOKENS, "{", "}", " "))</rewrite></seq>
       <seq><rewrite>transform((TOKENS, "{", "}", " ; "))</rewrite></seq>
       <seq><rewrite>transform((TOKENS, "{", "}", "AND"))</rewrite></seq>
       <seq><rewrite>transform((TOKENS, "{", "}", "ACCUM"))</rewrite></seq>
     </progression>
   </textquery>
  <score datatype="INTEGER" algorithm="COUNT"/>
</query>
```

The operator TRANSFORM is used to specify the rewrite rules and has the following syntax (note that it uses double parentheses). The parameters are described in Table 1–3.

```
TRANSFORM((terms, prefix, suffix, connector))
```

Table 1–3 TRANSFORM Parameters

Parameter	Description
term	Specifies the type of terms to be produced from the original query. Specify either TOKENS or THEMES. Specifying THEMES requires an installed knowledge base. A knowledge base may or may not have been installed with Oracle Text. For more information, see *Oracle Text Application Developer's Guide*.
prefix	Specifies the literal string to be prepended to all terms.
suffix	Specifies the literal string to be appended to all terms.
connector	Specifies the literal string to connect all terms after applying the prefix and suffix.

> **Note:** An error will be raised if the input Text query string specified in the Query Rewrite Template with TRANSFORM rules contains any Oracle Text query operators (such as AND, OR, or SOUNDEX). Also, any special characters (such as % or $) in the input Text query string must be preceded by an escape character, or an error is raised.

Query Relaxation Template

Use this template to progressively relax your query. **Progressive relaxation** is when you increase recall by progressively issuing less restrictive versions of a query, so that your application can return an appropriate number of hits to the user.

For example, the query of *black pen* can be progressively relaxed to:

```
black pen
black NEAR pen
black AND pen
black ACCUM pen
```

This is done with the following template

```
<query>
   <textquery lang="ENGLISH" grammar="CONTEXT">
     <progression>
       <seq>black pen</seq>
       <seq>black NEAR pen</seq>
```

```
      <seq>black AND pen</seq>
      <seq>black ACCUM pen</seq>
    </progression>
  </textquery>
  <score datatype="INTEGER" algorithm="COUNT"/>
</query>
```

Alternate Grammar Template

Use this template to specify an alternate grammar, such as CONTEXT or CATSEARCH. Specifying an alternate grammar enables you to enter queries using different syntax and operators.

For example, with CATSEARCH, enter ABOUT queries using the CONTEXT grammar. Likewise with CONTAINS, enter logical queries using the simplified CATSEARCH syntax.

The phrase *'dog cat mouse'* is interpreted as a phrase in CONTAINS. However, with CATSEARCH, this is equivalent to an AND query of *'dog AND cat AND mouse'*. Specify that CONTAINS use the alternate grammar with the following template:

```
<query>
  <textquery grammar="CTXCAT">dog cat mouse</textquery>
  <score datatype="integer"/>
</query>
```

Alternate Language Template

Use this template to specify an alternate language:

```
<query><textquery lang="french">bon soir</textquery></query>
```

Alternative Scoring Template

Use this template to specify an alternative scoring algorithm.

The following example specifies that the query use the CONTEXT grammar and return integer scores using the COUNT algorithm. This algorithm returns a score as the number of query occurrences in the document.

```
<query>
     <textquery grammar="CONTEXT" lang="english"> mustang
     </textquery>
     <score datatype="INTEGER" algorithm="COUNT"/>
</query>
```

The following example uses the normalization_expr attribute to add SDATA(*price*) into the score returned by the query, and uses it as the final score:

```
<query>
    <textquery grammar="CONTEXT" lang="english">
        DEFINESCORE(dog, RELEVANCE) and  cat
    </textquery>
    <score  algorithm="COUNT"  normalization_expr ="doc_score+ SDATA(price)"/>
</query>
```

The normalization_expr attribute is used only with the alternate scoring template, and is an arithmetic expression that consists of:

- Arithmetic operators: + - * /. The operator precedence is the same as that for SQL operator precedence.

- Grouping operators: (). Parentheses can be used to alter the precedence of the arithmetic operators.
- Absolute function: ABS(*n*) returns the absolute value of *n*; where *n* is any expression that returns a number.
- Logarithmic function: LOG(*n*): returns the base-10 logarithmic value of *n*; where *n* is any expression that returns a number.
- Predefined components: The `doc_score` predefined component can be used to return the initial query score of a particular document.
- `SDATA` component: `SDATA`*(name)* returns the value of the `SDATA` with the specified name as the score.
  - Only `SDATA` with a `NUMBER` or `DATE` data type is allowed. An error is raised otherwise.
  - The *sdata* string and the `SDATA` name are case-insensitive.
  - Because an `SDATA` section value can be `NULL`, any expression with `NULL` `SDATA` section value is evaluated as 0. For example: the `normalization_expr "doc_score + SDATA(price)"` will be evaluated to 0 if `SDATA(price)` for a given document has a `NULL` value.
- Numeric literals: There are any number literal that conforms to the SQL pattern of `NUMBER` literal and is within the range of the double-precision floating-point (`-3.4e38` to `3.4e38`).
- Date literals: Date literals must be enclosed with `DATE` (). Only the following format is allowed: `YYYY-MM-DD` or `YYYY-MM-DD HH24:MI:SS`. For example: `DATE(2005-11-08).`

  Consistent with SQL, if no time is specified, then `00:00:00` is assumed.

The `normalization_expr` attribute overrides the algorithm attribute. That is, if algorithm is set to `COUNT`, and the user also specifies `normalization_expr`, then the score will not be count, but the calculated score based on the `normalization_expr`.

If the score (either from `algorithm = COUNT` or `normalization_expr =` ...) is internally calculated to be greater than 100, then it will be set to 100.

If the query relaxation template is used, the score will be further normalized in such a way that documents returned from higher sequences will always have higher scores than documents returned from sequence(s) below.

DATE Literal Restrictions

Only the minus (-) operator is allowed between date-type data (`DATE` literals and date-type `SDATA`). Using other operators will result in an error. Subtracting two date-type data will produce a number (float) that represents the difference in number of days between the two dates. For example, the following expression is allowed:

```
SDATA(dob) -  DATE(2005-11-08)
```

The following expression is not allowed:

```
SDATA(dob) +  DATE(2005-11-08)
```

The plus (+) and minus (-) operators are allowed between numeric data and date type of data. The number operand is interpreted as the number or fraction of days. For example, the following expression is allowed:

```
DATE(2005-11-08) + 1        = 9 NOV 2005
```

The following expression is not allowed:

```
DATE(2005-11-08)* 3          = ERROR
```

Template Attribute Values

Table 1–4 gives the possible values for template attributes.

Table 1–4 Template Attribute Values

Tag Attribute	Description	Possible Values	Meaning
grammar=	Specifies the grammar of the query.	CONTEXT	The grammar of the query.
		CTXCAT	
datatype=	Specifies the type of number returned as score.	INTEGER	Returns score as integer between 0 and 100.
		FLOAT	Returns score as its high-precision floating-point number between 0 and 100.
algorithm=	Specifies the scoring algorithm to use.	DEFAULT	Returns the default.
		COUNT	Returns scores as the number of occurrences in the document.
lang=	Specifies the language name.	Any language supported by Oracle Database. See *Oracle Database Globalization Support Guide*.	The language name.

Template Grammar Definition

The query template interface is an XML document. Its grammar is defined with the following XML DTD:

```
<!ELEMENT query (textquery, score?)>
<!ELEMENT textquery (#PCDATA|progression)*>
<!ELEMENT progression (seq)+>
<!ELEMENT seq (#PCDATA|rewrite)*>
<!ELEMENT rewrite (#PCDATA)>
<!ELEMENT score EMPTY>
<!ATTLIST textquery grammar (context | ctxcat) #IMPLIED>
<!ATTLIST textquery language CDATA #IMPLIED>
<!ATTLIST score datatype (integer | float) "integer">
<!ATTLIST score algorithm (default | count) "default">
```

All tags and attributes values are case-sensitive.

> **See Also:** Chapter 3, "Oracle Text CONTAINS Query Operators" for more information about the operators in query expressions

label

Optionally, specifies the label that identifies the score generated by the CONTAINS operator.

Returns

For each row selected, the CONTAINS operator returns a number between 0 and 100 that indicates how relevant the document row is to the query. The number 0 means that Oracle Text found no matches in the row.

Note: You must use the SCORE operator with a label to obtain this number.

Example

The following example searches for all documents in the text column that contain the word *oracle*. The score for each row is selected with the SCORE operator using a label of 1:

```
SELECT SCORE(1), title from newsindex
    WHERE CONTAINS(text, 'oracle', 1) > 0;
```

The CONTAINS operator must be followed by an expression such as > 0, which specifies that the score value calculated must be greater than zero for the row to be selected.

When the SCORE operator is called (for example, in a SELECT clause), the CONTAINS clause must reference the score label value as in the following example:

```
SELECT SCORE(1), title from newsindex
     WHERE CONTAINS(text, 'oracle', 1) > 0 ORDER BY SCORE(1) DESC;
```

The following example specifies that the query be parsed using the CATSEARCH grammar:

```
SELECT id FROM test WHERE CONTAINS (text,
 '<query>
   <textquery lang="ENGLISH" grammar="CATSEARCH">
      cheap pokemon
   </textquery>
   <score datatype="INTEGER"/>
  </query>' ) > 0;
```

Grammar Template Example

The following example shows how to use the CTXCAT grammar in a CONTAINS query. The example creates a CTXCAT and a CONTEXT index on the same table, and compares the query results.

```
PROMPT create context and ctxcat indexes, both using theme indexing
PROMPT
create index tdrbqcq101x on test(text) indextype is ctxsys.context
parameters ('lexer theme_lexer');

create index tdrbqcq101cx on test(text) indextype is ctxsys.ctxcat
parameters ('lexer theme_lexer');

PROMPT *****  San Diego             ***********
PROMPT *****  CONTEXT grammar       ***********
PROMPT ** should be interpreted as phrase query **
select pk||' ==> '||text from test
where contains(text,'San Diego')>0
order by pk;

PROMPT *****  San Diego      ***********
PROMPT *****  CTXCAT grammar ***********
PROMPT ** should be interpreted as AND query  ***
select pk||' ==> '||text from test
where contains(text,
'<query>
  <textquery grammar="CTXCAT">San Diego</textquery>
  <score datatype="integer"/>
```

```
</query>')>0
order by pk;

PROMPT *****  Hitlist from CTXCAT index ***********
select pk||' ==> '||text from test
where catsearch(text,'San Diego','')>0
order by pk;
```

Alternate Scoring Query Template Example

The following query template adds price SDATA section (or SDATA filter-by column) value into the score returned by the query and uses it as the final score:

```
<query>
      <textquery grammar="CONTEXT" lang="english">
             DEFINESCORE(dog, RELEVANCE) and cat
      </textquery>
      <score algorithm="COUNT" normalization_expr ="doc_score+SDATA(price)"/>
</query>
```

Query Relaxation Template Example

The following query template defines a query relaxation sequence. The query of *black pen* is entered in sequence as *black pen*, then *black NEAR pen*, then *black AND pen*, and then *black ACCUM pen*. Query hits are returned in this sequence with no duplication as long as the application requires results.

```
select id from docs where CONTAINS (text, '
<query>
   <textquery lang="ENGLISH" grammar="CONTEXT">
     <progression>
       <seq>black pen</seq>
       <seq>black NEAR pen</seq>
       <seq>black AND pen</seq>
       <seq>black ACCUM pen</seq>
     </progression>
   </textquery>
   <score datatype="INTEGER" algorithm="COUNT"/>
</query>')>0;
```

Query relaxation is most effective when your application requires the top *n* hits to a query, which you can obtain with the DOMAIN_INDEX_SORT or FIRST_ROWS hint, which is being deprecated, in a PL/SQL cursor.

Query Rewrite Example

The following template defines a query rewrite sequence. The query of *kukui nut* is rewritten as follows:

{kukui} {nut}

{kukui} ; {nut}

{kukui} AND {nut}

{kukui} ACCUM {nut}

```
select id from docs where CONTAINS (text, '
 <query>
   <textquery lang="ENGLISH" grammar="CONTEXT"> kukui nut
     <progression>
       <seq><rewrite>transform((TOKENS, "{", "}", " "))</rewrite></seq>
       <seq><rewrite>transform((TOKENS, "{", "}", " ; "))</rewrite>/seq>
```

```
      <seq><rewrite>transform((TOKENS, "{", "}", "AND"))</rewrite><seq/>
      <seq><rewrite>transform((TOKENS, "{", "}", "ACCUM"))</rewrite><seq/>
    </progression>
  </textquery>
 <score datatype="INTEGER" algorithm="COUNT"/>
</query>')>0;
```

Notes

Querying Multilanguage Tables

With the multilexer preference, you can create indexes from multilanguage tables. At query time, the multilexer examines the session's language setting and uses the sublexer preference for that language to parse the query. If the language setting is not mapped, then the default lexer is used.

When the language setting is mapped, the query is parsed and run as usual. The index contains tokens from multiple languages, so such a query can return documents in several languages.

To limit your query to returning documents of a given language, use a structured clause on the language column.

Query Performance Limitation with a Partitioned Index

Oracle Text supports the `CONTEXT` indexing and querying of a partitioned text table.

However, for optimal performance when querying a partitioned table with an `ORDER BY SCORE` clause, query the partition. If you query the entire table and use an `ORDER BY SCORE` clause, the query might not perform optimally unless you include a range predicate that can limit the query to a single partition.

For example, the following statement queries the partition `p_tab4` partition directly:

```
select * from part_tab partition (p_tab4) where contains(b,'oracle') > 0 ORDER BY
SCORE DESC;
```

Related Topics

"Syntax for CONTEXT Index Type" on page 1-37

Chapter 3, "Oracle Text CONTAINS Query Operators"

"The CONTEXT Grammar" topic in *Oracle Text Application Developer's Guide*

"SCORE" on page 1-62

CREATE INDEX

This section describes the CREATE INDEX statement as it pertains to creating an Oracle Text domain index and composite domain index.

> **See Also:** "*Oracle Database SQL Language Reference* for a complete description of the CREATE INDEX statement

Purpose

Use CREATE INDEX to create an Oracle Text index. An Oracle Text index is an Oracle Database domain index or composite domain index of type CONTEXT, CTXCAT, CTXRULE, or CTXXPATH. A domain index is an application-specific index. A composite domain index (CDI) is an Oracle Text index that not only indexes and processes a specified text column, but also indexes and processes FILTER BY and ORDER BY structured columns, which are specified during index creation.

You must create an appropriate Oracle Text index to enter CONTAINS, CATSEARCH, or MATCHES queries.

You cannot create an Oracle Text index on an index-organized table.

You can create the following types of Oracle Text indexes.

CONTEXT

A CONTEXT index is the basic type of Oracle Text index. This is an index on a text column. A CONTEXT index is useful when your source text consists of many large, coherent documents. Query this index with the CONTAINS operator in the WHERE clause of a SELECT statement. This index requires manual synchronization after DML. See "Syntax for CONTEXT Index Type" on page 1-37.

CTXCAT

The CTXCAT index is a combined index on a text column and one or more other columns. The CTXCAT type is typically used to index small documents or text fragments, such as item names, prices, and descriptions found in catalogs. Query this index with the CATSEARCH operator in the WHERE clause of a SELECT statement. This type of index is optimized for mixed queries. This index is transactional, automatically updating itself with DML to the base table. See "Syntax for CTXCAT Index Type" on page 1-52.

CTXRULE

A CTXRULE index is used to build a document classification application. The CTXRULE index is an index created on a table of queries or a column containing a set of queries, where the queries serve as rules to define the classification criteria. Query this index with the MATCHES operator in the WHERE clause of a SELECT statement. See "Syntax for CTXRULE Index Type" on page 1-55.

CTXXPATH

The CTXPATH index is used to speed up existsNode() queries on an XMLType column. See "Syntax for CTXXPATH Index Type" on page 1-56.

Required Privileges

You do not need the CTXAPP role to create an Oracle Text index. If you have Oracle Database grants to create a B-tree index on the text column, you have sufficient privilege to create a text index. The issuing owner, table owner, and index owner can all be different users, which is consistent with Oracle standards for creating regular B-tree indexes.

Syntax for CONTEXT Index Type

Uses a CONTEXT index to create an index on a text column. Query this index with the CONTAINS operator in the WHERE clause of a SELECT statement. This index requires manual synchronization after DML.

```
CREATE INDEX [schema.]index ON [schema.]table(txt_column)
  INDEXTYPE IS ctxsys.context [ONLINE]
  [FILTER BY filter_column[, filter_column]...]
  [ORDER BY oby_column[desc|asc][, oby_column[desc|asc]]...]
  [LOCAL [(PARTITION [partition] [PARAMETERS('paramstring')]
  [, PARTITION [partition] [PARAMETERS('paramstring')]])]
  [PARAMETERS(paramstring)] [PARALLEL n] [UNUSABLE]];
```

[*schema*.]*index*
Specifies the name of the Text index to create.

[*schema*.]*table*(*txt_column*)
Specifies the name of the table and column to index. txt_column is the name of the domain index column on which the CONTAINS() operator will be invoked.

Your table can optionally contain a primary key if you prefer to identify your rows as such when you use procedures in CTX_DOC. When your table has no primary key, document services identifies your documents by ROWID.

The column that you specify must be one of the following types: CHAR, VARCHAR, VARCHAR2, BLOB, CLOB, BFILE, XMLType, or URIType.

The table that you specify can be a partitioned table. If you do not specify the LOCAL clause, then a global, nonpartitioned index is created.

The DATE, NUMBER, and nested table columns cannot be indexed. Object columns also cannot be indexed, but their attributes can be indexed, provided that they are atomic data types.

Attempting to create an index on a Virtual Private Database (VPD) protected table will fail unless one of the following criteria is true:

- The VPD policy is created such that it does not apply to the INDEX statement type.
- The policy function returns a NULL predicate for the current user.
- The user (or index owner) is SYS.
- The user has the EXEMPT ACCESS POLICY privilege.

Indexes on multiple columns are not supported with the CONTEXT index type. You must specify only one column in the column list.

> **Note:** With the CTXCAT index type, you can create indexes on text and structured columns. See "Syntax for CTXCAT Index Type" on page 1-52

Note: Because a transparent data encryption-enabled column does not support domain indexes, it cannot be used with Oracle Text. However, you can create an Oracle Text index on a column in a table stored in a TDE-enabled tablespace.

ONLINE

Creates the index while enabling DML insertions/updates/deletions on the base table.

During indexing, Oracle Text enqueues DML requests in a pending queue. At the end of the index creation, Oracle Text locks the base table. During this time, DML is blocked. You must synchronize the index in order for DML changes to be available.

Limitations

The following limitations apply to using `ONLINE`:

- At the very beginning or very end of the `ONLINE` process, DML might fail.
- `ONLINE` is supported for `CONTEXT` indexes only.

FILTER BY filter_column

This is the structured indexed column on which a range or equality predicate in the `WHERE` clause of a mixed query will operate. You can specify one or more structured columns for `filter_column`, on which the relational predicates are expected to be specified along with the `CONTAINS()` predicate in a query.

The cost-based optimizer (CBO) will consider pushing down the structured predicates on these `FILTER BY` columns with the following relational operators: `<`, `<=`, `=`, `>=`, `>`, `between`, and `LIKE` (for `VARCHAR2`).

These columns can only be of `CHAR`, `NUMBER`, `DATE`, `VARCHAR2`, or `RAW` type. Additionally, `VARCHAR2` and `RAW` types are supported only if the maximum length is specified and is limited to no more than `249`. The `ADT` attributes of supported types (`CHAR`, `NUMBER`, `DATE`, `VARCHAR2`, or `RAW`) are also allowed. An error is raised for all other data types. Expressions, for example, `func(cola)`, and virtual columns are not allowed.

`txt_column` is allowed in the `FILTER BY` column list.

DML operations on `FILTER BY` columns are always transactional.

ORDER BY oby_column

This is the structured indexed column on which a structured `ORDER BY` mixed query will be based. A list of structured *oby_columns* can be specified in the `ORDER BY` clause of a `CONTAINS()` query.

These columns can only be of `CHAR`, `NUMBER`, `DATE`, `VARCHAR2`, or `RAW` type. Additionally, `VARCHAR2` and `RAW` types are supported only if the maximum length is specified and is limited to no more than `249`. Expressions, for example, `func(cola)`, and virtual columns are not allowed.

The order of the specified columns matters. The cost-based optimizer will consider pushing the sort into the composite domain index only if the `ORDER BY` clause in the text query contains:

- Entire ordered `ORDER BY` columns declared by the `ORDER BY` clause during `CREATE INDEX`,
- CBO will consider pushing the sort into the CDI only if the `ORDER BY` clause in the text query contains:

Entire ordered ORDER BY columns declared by the ORDER BY clause during the CREATE INDEX statement
Only the prefix of the ordered ORDER BY columns declared by the ORDER BY clause during the CREATE INDEX statement
The score followed by the prefix of the ordered ORDER BY columns declared by the ORDER BY clause during the CREATE INDEX statement
The score following the prefix of the ordered ORDER BY columns declared by the ORDER BY clause during the CREATE INDEX statement

The following example illustrates CBO behavior with regard to ORDER BY columns:

```
CREATE INDEX foox ON foo(D) INDEXTYPE IS CTXSYS.CONTEXT
FILTER BY B, C
ORDER BY A, B desc;
```

Consider the following query:

```
SELECT A, SCORE(1) FROM foo WHERE CONTAINS(D, 'oracle',1)>0
AND C>100 ORDER BY col_list;
```

Note: If you set NLS_SORT or NLS_COMP parameters (that is, `alter session set NLS_SORT = <some lang>;`), then CBO will not push the sort or related structured predicate into the CDI. This behavior is consistent with regular B-tree indexes.

CBO will consider pushing the sort into CDI if col_list has the following values:

```
A
A, B desc
SCORE(1), A
SCORE(1), A, B desc
A, SCORE(1)
A, B desc, SCORE(1)
```

CBO will not consider to push the sort into CDI if col_list has the following values:

```
B
B,A
SCORE(1), B
B, SCORE(1)
A, B, C
A, B asc (or simply A, B)
```
(or simply A, B)

- score followed by the prefix of the ordered ORDER BY columns declared by the ORDER BY clause during the CREATE INDEX statement.
- The score following the prefix of the ordered ORDER BY columns declared by the ORDER BY clause during the CREATE INDEX statement.

Expressions, for example, func(cola), are not allowed.

txt_column appearing in the ORDER BY column list is allowed.

DML operations on ORDER BY columns are always transactional.

Limitations

The following limitations apply to FILTER BY and ORDER BY:

- A structured column is allowed in FILTER BY and ORDER BY clauses. However, a column that is mapped to MDATA in a FILTER BY clause cannot also appear in the ORDER BY clause. An error will be raised in this case.
- The maximum length for CHAR, VARCHAR2, and RAW columns cannot be greater than 249. Additionally, if the VARCHAR2 or RAW column is mapped to an MDATA column, then the specified maximum length cannot exceed 64 and 32 bytes, respectively. (Note that MDATA does not support CHAR data types. If a FILTER BY column of CHAR data type is mapped to an MDATA section, then an error will be raised during the CREATE INDEX statement.)
- The sum of the numbers for INDEXED_COLUMN, FILTER BY columns, and ORDER BY columns cannot be greater than 32.

Note:

- As with concatenated B-tree indexes or bitmap indexes, performance degradation may occur in DML as the number of FILTER BY and ORDER BY columns increases.
- Mapping a FILTER BY column to MDATA is not recommended if the FILTER BY column contains sequential values or has very high cardinality. Doing so can result in a very long and narrow $I table and reduced $X performance. An example is a column of type DATE. For columns of this type, mapping to SDATA is recommended.

Note: An index table with the name DR$*indextable*$S is created to store FILTER BY and ORDER BY columns that are mapped to SDATA sections. If nothing is mapped to an SDATA section, then the $S table will not be created.

$S table contains the following columns:

- SDATA_ID *number* is the internal SDATA section ID.
- SDATA_LAST *number*, the last document ID, which is analogous to token_last.
- SDATA_DATA RAW(2000), the compressed SDATA values. Note that if $S is created on a tablespace with 4K database block size, then it will be defined as RAW(1500).

Restriction: For performance reasons, $S table must be created on a tablespace with db block size >= 4K without overflow segment and without PCTTHRESHOLD clause. If $S is created on a tablespace with db block size < 4K, or is created with an overflow segment or with a PCTTHRESHOLD clause, then appropriate errors will be raised during the CREATE INDEX statement.

Restrictions on exporting and importing text tables with composite domain index created with FILTER BY and/or ORDER BY clauses are as follows:

- Regular exp and imp will not support exporting and importing of composite domain index. Doing so will lead to the following error: EXP-00113: Feature Composite Domain Index is unsupported.
- To export a text table with composite domain index, you must use Data Pump Export and Import utilities (invoked with the expdp and impdp commands, respectively), or DBMS_DATAPUMP PL/SQL package.

> **See Also:** ADD_SDATA_COLUMN in Chapter 7, "CTX_DDL Package"

Limitations of using ALTER INDEX and ALTER TABLE with FILTER BY and ORDER BY columns of the composite domain index, which are imposed by Extensible Indexing Framework in Oracle Database:

(These limitations are imposed by Extensible Indexing Framework in Oracle Database.)

- Using ALTER INDEX to add or drop FILTER BY and ORDER BY columns is currently not supported. You must re-create the index to add or drop FILTER BY or ORDER BY columns.
- To use ALTER TABLE MODIFY COLUMN to modify the datatype of a column that has the composite domain index built on it, you must first drop the composite domain index before modifying the column.
- To use ALTER TABLE DROP COLUMN to drop a column that is part of the composite domain index, you must first drop the composite domain index before dropping the index column.

The following limitations apply to FILTER BY and ORDER BY when used with PL/SQL packages:

- Mapping FILTER BY columns to sections is optional. If section mapping does not exist for a FILTER BY column, then it is mapped to an SDATA section by default. The section name assumes the name of the FILTER BY column.
- If a section group is not specified during the CREATE INDEX clause of a composite domain index, then system default section group settings are used. An SDATA section is created for each of the FILTER BY and ORDER BY columns.

> **Note:** Because a section name does not allow certain special characters and is case-insensitive, if the column name is case-sensitive or contains special characters, then an error will be raised. To work around this problem, you must map the column to an MDATA or SDATA section before creating the index. See CTX_DDL.ADD_MDATA_COLUMN or CTX_DDL.ADD_SDATA_COLUMN.

- An error is raised if a column that is mapped to an MDATA section also appears in the ORDER BY column clause.
- Column section names are unique to their section group. That is, you cannot have an MDATA column section named FOO if you already have an MDATA column section named FOO. Nor can you have a field section named FOO if you already have an SDATA column section named FOO. This is true whether it is implicitly created (by CREATE INDEX for FILTER BY or ORDER BY clauses) or explicitly created (by CTX_DDL.ADD_SDATA_COLUMN).

- One section name can be mapped to only one FILTER BY column, and vice versa. Mapping a section to more than one column, or mapping a column to more than one section is not allowed.
- Column sections can be added to any type of section group, including the NULL section group.
- If a section group with sections added by the CTX_DDL.ADD_MDATA_COLUMN or CTX_DDL.ADD_SDATA_COLUMN packages is specified for a CREATE INDEX statement without a FILTER BY clause, then the mapped column sections will be ignored. However, the index will still get created without those column sections. The same is true for a FILTER BY clause that does not contain mapped columns in the specified section group.

See Also: CTX_DDL.ADD_SDATA_COLUMN

LOCAL [(PARTITION [*partition*] [PARAMETERS('*paramstring*')]
Specifies a local partitioned context index on a partitioned table. The partitioned table must be partitioned by range. Hash, composite, and list partitions are not supported.

You can specify the list of index partition names with *partition_name*. If you do not specify a partition name, then the system assigns one. The order of the index partition list must correspond to the table partition order.

The PARAMETERS clause associated with each partition specifies the parameters string specific to that partition. You can only specify *sync (manual | every | on commit)*, *memory* and *storage* for each index partition.

The PARAMETERS clause also supports the POPULATE and NOPOPULATE arguments. See "POPULATE | NOPOPULATE" on page 1-46.

Query the views CTX_INDEX_PARTITIONS or CTX_USER_INDEX_PARTITIONS to find out index partition information, such as index partition name, and index partition status.

See Also: "Creating a Local Partitioned Index" on page 1-49

Query Performance Limitation with Partitioned Index

For optimal performance when querying a partitioned index with an ORDER BY SCORE clause, query the partition. If you query the entire table and use an ORDER BY SCORE clause, the query might not perform optimally unless you include a range predicate that can limit the query to the fewest number of partitions, which is optimally a single partition.

See Also: "Query Performance Limitation with a Partitioned Index" on page 1-35

PARALLEL n
Optionally specifies the parallel degree for parallel indexing. The actual degree of parallelism might be smaller depending on your resources. You can use this parameter on nonpartitioned tables. However, creating a nonpartitioned index in parallel does not turn on parallel query processing. Parallel indexing is supported for creating a local partitioned index.

The indexing memory size specified in the parameter clause applies to each parallel slave. For example, if indexing memory size is specified in the parameter clause as 500M and parallel degree is specified as 2, then you must ensure that there is at least 1GB of memory available for indexing.

See Also:

"Parallel Indexing" on page 1-50

"Creating a Local Partitioned Index in Parallel" on page 1-50

The "Performance Tuning" chapter in *Oracle Text Application Developer's Guide*

Performance

Parallel indexing can speed up indexing when you have large amounts of data to index and when your operating system supports multiple CPUs.

Note: Using `PARALLEL` to create a local partitioned index that enables parallel queries. (Creating a nonpartitioned index in parallel does not turn on parallel query processing.)

Parallel querying degrades query throughput especially on heavily loaded systems. Because of this, Oracle recommends that you disable parallel querying after creating a local index. To do so, use the `ALTER INDEX NOPARALLEL` statement.

For more information on parallel querying, see the "Performance Tuning" chapter in *Oracle Text Application Developer's Guide*.

Limitations

Parallel indexing is supported only for the `CONTEXT` index type.

UNUSABLE

Creates an unusable index. This creates index metadata only and exits immediately.

You might create an unusable index when you need to create a local partitioned index in parallel.

See Also: "Creating a Local Partitioned Index in Parallel" on page 1-50

PARAMETERS(*paramstring*)

Optionally specify indexing parameters in `paramstring`. You can specify preferences owned by another user using the `user.preference` notation.

The syntax for `paramstring` is as follows:

```
paramstring =
'[DATASTORE datastore_pref]
 [FILTER filter_pref]
 [CHARSET COLUMN charset_column_name]
 [FORMAT COLUMN format_column_name]

 [LEXER lexer_pref]
 [LANGUAGE COLUMN language_column_name]

 [WORDLIST wordlist_pref]
 [STORAGE storage_pref]
 [STOPLIST stoplist]
 [SECTION GROUP section_group]
 [MEMORY memsize]
 [POPULATE | NOPOPULATE]
 [SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)]
 [TRANSACTIONAL]'
```

Create datastore, filter, lexer, wordlist, and storage preferences with CTX_DDL.CREATE_PREFERENCE and then specify them in the paramstring.

Note: When you specify no paramstring, Oracle Text uses the system defaults.

For more information about these defaults, see "Default Index Parameters" on page 2-77.

DATASTORE *datastore_pref*
Specifies the name of your datastore preference. Use the datastore preference to specify where your text is stored.See "Datastore Types" on page 2-2.

FILTER *filter_pref*
Specifies the name of your filter preference. Use the filter preference to specify how to filter formatted documents to plain text or HTML. See "Filter Types" on page 2-16.

CHARSET COLUMN *charset_column_name*
Specifies the name of the character set column. This column must be in the same table as the text column, and it must be of type CHAR, VARCHAR, or VARCHAR2. Use this column to specify the document character set for conversion to the database character set. The value is case-insensitive. You must specify a globalization support character set string, such as JA16EUC.

When the document is plain text or HTML, the AUTO_FILTER and CHARSET filters use this column to convert the document character set to the database character set for indexing.

Use this column when you have plain text or HTML documents with different character sets or in a character set different from the database character set.

Setting NLS_LENGTH_SEMANTICS parameter to CHAR is not supported at the database level. This parameter is supported for the following columns:

- The CHARSET COLUMN, for example:

```
VARCHAR2 <size> CHAR
CHAR <size> CHAR
```

- An index created on a VARCHAR2 and CHAR column
- VARCHAR2 and CHAR columns for FILTER BY and ORDER BY clauses of CREATE INDEX
- FORMAT COLUMN

Note:

Documents are not marked for re-indexing when only the character set column changes. The indexed column must be updated to flag the re-index.

The NLS_LENGTH_SEMANTICS = CHAR parameter is supported at the column level only, and is not supported at the database level, as described in this section.

FORMAT COLUMN *format_column_name*
Specifies the name of the format column. The format column must be in the same table as the text column and it must be CHAR, VARCHAR, or VARCHAR2 type.

FORMAT COLUMN determines how a document is filtered, or, in the case of the IGNORE value, if it is to be indexed.

AUTO_FILTER uses the format column when filtering documents. Use this column with heterogeneous document sets to optionally bypass filtering for plain text or HTML documents.

In the format column, you can specify one of the following options:

- TEXT
- BINARY
- IGNORE

The TEXT option indicates that the document is either plain text or HTML. When TEXT is specified, the document is not filtered, but may have the character set converted.

The BINARY option indicates that the document is a format supported by the AUTO_FILTER object other than plain text or HTML, for example PDF. BINARY is the default, if the format column entry cannot be mapped.

The IGNORE option indicates that the row is to be ignored during indexing. Use this value when you need to bypass rows that contain data incompatible with text indexing such as image data, or rows in languages that you do not want to process. The difference between documents with TEXT and IGNORE format column types is that the former are indexed but ignored by the filter, while the latter are not indexed at all. Thus, IGNORE can be used with any filter type.

Note: Documents are not marked for re-indexing when only the format column changes. The indexed column must be updated to flag the re-index.

LEXER *lexer_pref*
Specifies the name of your lexer or multilexer preference. Use the lexer preference to identify the language of your text and how text is tokenized for indexing. See "Lexer Types" on page 2-30.

LANGUAGE COLUMN *language_column_name*
Specifies the name of the language column when using a multi-lexer preference. See "MULTI_LEXER" on page 2-38.

This column must exist in the base table. It cannot be the same column as the indexed column. Only the first 30 bytes of the language column are examined for language identification.

Note: Documents are not marked for re-indexing when only the language column changes. The indexed column must be updated to flag the re-index.

WORDLIST *wordlist_pref*
Specifies the name of your wordlist preference. Use the wordlist preference to enable features such as fuzzy, stemming, and prefix indexing for better wildcard searching. See "Wordlist Type" on page 2-57.

STORAGE *storage_pref*
Specifies the name of your storage preference for the Text index. Use the storage preference to specify how the index tables are stored. See "Storage Types" on page 2-64.

STOPLIST *stoplist*
Specifies the name of your stoplist. Use stoplist to identify words that are not to be indexed. See CTX_DDL.CREATE_STOPLIST.

SECTION GROUP *section_group*
Specifies the name of your section group. Use section groups to create searchable sections in structured documents. See CTX_DDL.CREATE_SECTION_GROUP.

MEMORY *memsize*
Specifies the amount of run-time memory to use for indexing. The syntax for memsize is as follows:

```
memsize = number[K|M|G]
```

K stands for kilobytes, M stands for megabytes, and G stands for gigabytes.

The value you specify for `memsize` must be between 1M and the value of `MAX_INDEX_MEMORY` in the CTX_PARAMETERS view. To specify a memory size larger than the `MAX_INDEX_MEMORY`, you must reset this parameter with CTX_ADM.SET_PARAMETER to be larger than or equal to `memsize`.

The default is the value specified for `DEFAULT_INDEX_MEMORY` in `CTX_PARAMETERS`.

The `memsize` parameter specifies the amount of memory Oracle Text uses for indexing before flushing the index to disk. Specifying a large amount memory improves indexing performance because there are fewer I/O operations and improves query performance and maintenance, because there is less fragmentation.

Specifying smaller amounts of memory increases disk I/O and index fragmentation, but might be useful when run-time memory is scarce.

POPULATE | NOPOPULATE
Specifies whether an index should be empty or populated. The default is `POPULATE`.

Note: `POPULATE | NOPOPULATE` is the only option whose default value cannot be set with CTX_ADM.SET_PARAMETER.

This option is not valid with `CTXXPATH` indexes.

Empty indexes are populated by updates or inserts to the base table. You might create an empty index when you need to create your index incrementally or to selectively index documents in the base table. You might also create an empty index when you require only theme and Gist output from a document set.

SYNC (MANUAL | EVERY "*interval-string*" | ON COMMIT)
Specifies `SYNC` for automatic synchronization of the `CONTEXT` index when there are inserts, updates or deletes to the base table. You can specify one of the following `SYNC` methods:

Table 1–5 SYNC Types

SYNC Type	Description
MANUAL	Provides no automatic synchronization. This is the default. You must manually synchronize the index with CTX_DDL.SYNC_INDEX.
EVERY "*interval-string*"	Automatically synchronizes the index at a regular interval specified by the value of *interval-string*, which takes the same syntax as that for scheduler jobs. Automatic synchronization using EVERY requires that the index creator have CREATE JOB privileges. Ensure that *interval-string* is set to a long enough period that any previous sync jobs will have completed; otherwise, the sync job might hang. *interval-string* must be enclosed in double quotes, and any single quote within *interval-string* must be preceded by the escape character with another single quote. See "Enabling Automatic Index Synchronization" on page 1-48 for an example of automatic sync syntax.
ON COMMIT	Synchronizes the index immediately after a commit transaction. The commit transaction does not return until the sync is complete. (Because the synchronization is performed as a separate transaction, there may be a period, usually small, when the data is committed but index changes are not.) The operation uses the memory specified with the *memory* parameter. Note that the sync operation has its own transaction context. If this operation fails, the data transaction is still commited. Index synchronization errors are logged in the CTX_USER_INDEX_ERRORS view. See "Viewing Index Errors" on page 1-52. See "Enabling Automatic Index Synchronization" on page 1-48 for an example of ON COMMIT syntax.

Each partition of a locally partitioned index can have its own type of sync (ON COMMIT, EVERY, or MANUAL). The type of sync specified in master parameter strings applies to all index partitions unless a partition specifies its own type.

With automatic (EVERY) synchronization, users can specify memory size and parallel synchronization. That syntax is:

```
... EVERY interval_string MEMORY mem_size PARALLEL paradegree ...
```

The ON COMMIT synchronizations can be run only serially and must use the same memory size that was specified at index creation.

> **See Also:** *Oracle Database Administrator's Guide* for information about job scheduling

TRANSACTIONAL

Specifies that documents can be searched immediately after they are inserted or updated. If a text index is created with TRANSACTIONAL enabled, then, in addition to processing the synchronized rowids already in the index, the CONTAINS operator will process unsynchronized rowids as well. Oracle Text does in-memory indexing of unsynchronized rowids and processes the query against the in-memory index.

TRANSACTIONAL is an index-level parameter and does not apply at the partition level.

You must still synchronize your text indexes from time to time (with CTX_DDL.SYNC_INDEX) to bring pending rowids into the index. Query performance degrades as the

number of unsynchronized rowids increases. For that reason, Oracle recommends setting up your index to use automatic synchronization with the EVERY parameter. (See "SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)" on page 1-46.)

Transactional querying for indexes that have been created with the TRANSACTIONAL parameter can be turned on and off (for the duration of a user session) with the PL/SQL variable CTX_QUERY.disable_transactional_query. This is useful, for example, if you find that querying is slow due to the presence of too many pending rowids. Here is an example of setting this session variable:

```
exec ctx_query.disable_transactional_query := TRUE;
```

If the index uses AUTO_FILTER, queries involving unsynchronized rowids will require filtering of unsynchronized documents.

CREATE INDEX: CONTEXT Index Examples

The following sections give examples of creating a CONTEXT index.

Creating CONTEXT Index Using Default Preferences

The following example creates a CONTEXT index called myindex on the docs column in mytable. Default preferences are used.

```
CREATE INDEX myindex ON mytable(docs) INDEXTYPE IS ctxsys.context;
```

See Also:

- *Oracle Text Application Developer's Guide*
- For more information about default settings, see "Default Index Parameters" on page 2-77

Creating CONTEXT Index with Custom Preferences

The following example creates a CONTEXT index called myindex on the docs column in mytable. The index is created with a custom lexer preference called my_lexer and a custom stoplist called my_stop.

This example also assumes that the preference and stoplist were previously created with CTX_DDL.CREATE_PREFERENCE for my_lexer, and CTX_DDL.CREATE_STOPLIST for my_stop. Default preferences are used for the unspecified preferences.

```
CREATE INDEX myindex ON mytable(docs) INDEXTYPE IS ctxsys.context
  PARAMETERS('LEXER my_lexer STOPLIST my_stop');
```

Any user can use any preference. To specify preferences that exist in another user's schema, add the user name to the preference name. The following example assumes that the preferences my_lexer and my_stop exist in the schema that belongs to user kenny:

```
CREATE INDEX myindex ON mytable(docs) INDEXTYPE IS ctxsys.context
  PARAMETERS('LEXER kenny.my_lexer STOPLIST kenny.my_stop');
```

Enabling Automatic Index Synchronization

You can create your index and specify that the index be synchronized at regular intervals for insertions, updates and deletions to the base table. To do so, create the index with the SYNC (EVERY "*interval-string*") parameter.

To use job scheduling, you must log in as a user who has DBA privileges and then grant CREATE JOB privileges.

The following example creates an index and schedules three synchronization jobs for three index partitions. The first partition uses ON COMMIT synchronization. The other two partitions are synchronized by jobs that are scheduled to be executed every Monday at 3 P.M.

```
CONNECT system/manager
GRANT CREATE JOB TO dr_test

CREATE INDEX tdrmauto02x ON tdrmauto02(text)
   INDEXTYPE IS CTXSYS.CONTEXT local
   (PARTITION tdrm02x_i1 PARAMETERS('
   MEMORY 20m SYNC(ON COMMIT)'),
   PARTITION tdrm02x_i2,
   PARTITION tdrm02x_i3)  PARAMETERS('
   SYNC (EVERY "NEXT_DAY(TRUNC(SYSDATE), ''MONDAY'') + 15/24")
  ');
```

See *Oracle Database Administrator's Guide* for information about job scheduling syntax.

Creating CONTEXT Index with Multilexer Preference

The multilexer preference decides which lexer to use for each row based on a language column. This is a character column in the table that stores the language of the document in the text column. For example, create the table globaldoc to hold documents of different languages:

```
CREATE TABLE globaldoc (
   doc_id NUMBER PRIMARY KEY,
   lang VARCHAR2(10),
   text CLOB
);
```

Assume that global_lexer is a multilexer preference you created. To index the global_doc table, specify the multilexer preference and the name of the language column as follows:

```
CREATE INDEX globalx ON globaldoc(text) INDEXTYPE IS ctxsys.context PARAMETERS
('LEXER global_lexer LANGUAGE COLUMN lang');
```

> **See Also:** "MULTI_LEXER" on page 2-38 for more information about creating multilexer preferences

Creating a Local Partitioned Index

The following example creates a text table that is partitioned into three, populates it, and then creates a partitioned index:

```
PROMPT create partitioned table and populate it

CREATE TABLE part_tab (a int, b varchar2(40)) PARTITION BY RANGE(a)
(partition p_tab1 values less than (10),
 partition p_tab2 values less than (20),
 partition p_tab3 values less than (30));

PROMPT create partitioned index
CREATE INDEX part_idx on part_tab(b) INDEXTYPE IS CTXSYS.CONTEXT
LOCAL (partition p_idx1, partition p_idx2, partition p_idx3);
```

Note: The limit for the number of partitions in Oracle Text is the same as the maximum number of partitions per table in Oracle Database.

Using FILTER BY and ORDER BY Clauses

The following example creates an index on table *docs* and orders the documents by author's publishing date.

First, create the table:

```
CREATE TABLE docs (
    docid    NUMBER,
    pub_date DATE,
    author   VARCHAR2(30),
    category VARCHAR2(30),
    document CLOB
);
```

Create the index with `FILTER BY` and `ORDER BY` clauses:

```
CREATE INDEX doc_idx on docs(document) indextype is ctxsys.context
  FILTER BY category, author
  ORDER BY pub_date desc, docid
  PARAMETERS ('memory 500M');
```

Parallel Indexing

Parallel indexing can improve index performance when you have multiple CPUs.

To create an index in parallel, use the `PARALLEL` clause with a parallel degree. This example uses a parallel degree of 3:

```
CREATE INDEX myindex ON mytab(pk) INDEXTYPE IS ctxsys.context PARALLEL 3;
```

Creating a Local Partitioned Index in Parallel

Creating a local partitioned index in parallel can improve performance when you have multiple CPUs. With partitioned tables, you can divide the work. You can create a local partitioned index in parallel in two ways:

- Use the `PARALLEL` clause with the `LOCAL` clause in the `CREATE INDEX` statement. In this case, the maximum parallel degree is limited to the number of partitions you have. See "Parallelism with CREATE INDEX" on page 1-51.
- Create an unusable index first, then run the `DBMS_PCLXUTIL.BUILD_PART_INDEX` utility. This method can result in a higher degree of parallelism, especially if you have more CPUs than partitions. See "Parallelism with DBMS_PCLUTIL.BUILD_PART_INDEX" on page 1-51.

If you attempt to create a local partitioned index in parallel, and the attempt fails, you may see the following error message:

```
ORA-29953: error in the execution of the ODCIIndexCreate routine for one or more
of the index partitions
```

To determine the specific reason why the index creation failed, query the CTX_USER_INDEX_ERRORS view.

Parallelism with CREATE INDEX

You can achieve local index parallelism by using the PARALLEL and LOCAL clauses in the CREATE INDEX statement. In this case, the maximum parallel degree is limited to the number of partitions that you have.

The following example creates a table with three partitions, populates them, and then creates the local indexes in parallel with a degree of 2:

```
create table part_tab3(id number primary key, text varchar2(100))
partition by range(id)
(partition p1 values less than (1000),
 partition p2 values less than (2000),
 partition p3 values less than (3000));

begin
  for i in 0..2999
  loop
      insert into part_tab3 values (i,'oracle');
  end loop;
end;
/

create index part_tab3x on part_tab3(text)
indextype is ctxsys.context local (partition part_tabx1,
                                   partition part_tabx2,
                                   partition part_tabx3)
parallel 2;
```

Parallelism with DBMS_PCLUTIL.BUILD_PART_INDEX

You can achieve local index parallelism by first creating an unusable CONTEXT index, and then running the DBMS_PCLUTIL.BUILD_PART_INDEX utility. This method can result in a higher degree of parallelism, especially when you have more CPUs than partitions.

In this example, the base table has three partitions. We create a local partitioned unusable index first, then run DBMS_PCLUTIL.BUILD_PART_INDEX, which builds the 3 partitions in parallel (referred to as inter-partition parallelism). Also, inside each partition, index creation proceeds in parallel (called intra-partition parallelism) with a parallel degree of 2. Therefore, the total parallel degree is 6 (3 times 2).

```
create table part_tab3(id number primary key, text varchar2(100))
partition by range(id)
(partition p1 values less than (1000),
 partition p2 values less than (2000),
 partition p3 values less than (3000));

begin
  for i in 0..2999
  loop
      insert into part_tab3 values (i,'oracle');
  end loop;
end;
/

create index part_tab3x on part_tab3(text)
indextype is ctxsys.context local (partition part_tabx1,
                                   partition part_tabx2,
                                   partition part_tabx3)
unusable;
```

```
exec dbms_pclxutil.build_part_index(jobs_per_batch=>3,
  procs_per_job=>2,
  tab_name=>'PART_TAB3',
  idx_name=>'PART_TAB3X',
  force_opt=>TRUE);
```

Viewing Index Errors

After a CREATE INDEX or ALTER INDEX operation, you can view index errors with Oracle Text views. To view errors on your indexes, query the CTX_USER_INDEX_ERRORS view. To view errors on all indexes as CTXSYS, query the CTX_INDEX_ERRORS view.

For example, to view the most recent errors on your indexes, enter the following statement:

```
SELECT err_timestamp, err_text FROM ctx_user_index_errors
ORDER BY err_timestamp DESC;
```

Deleting Index Errors

To clear the index error view, enter the following statement:

```
DELETE FROM ctx_user_index_errors;
```

Syntax for CTXCAT Index Type

Combines an index on a text column and one or more other columns. Query this index with the CATSEARCH operator in the WHERE clause of a SELECT statement. This type of index is optimized for mixed queries. This index is transactional, automatically updating itself with DML to the base table.

```
CREATE INDEX [schema.]index on [schema.]table(column) INDEXTYPE IS ctxsys.ctxcat
[PARAMETERS
('[index set index_set]
[lexer lexer_pref]
[storage storage_pref]
[stoplist stoplist]
[section group sectiongroup_pref
[wordlist wordlist_pref]
[memory memsize]');
```

[*schema*.]*table*(*column*)
Specifies the name of the table and column to index.

The column that you specify when you create a CTXCAT index must be of type CHAR or VARCHAR2. No other types are supported for CTXCAT.

Attempting to create an index on a Virtual Private Database (VPD) protected table will fail unless one of the following options is true:

- The VPD policy is created such that it does not apply to INDEX statement type, which is the default
- The policy function returns a null predicate for the current user.
- The user (index owner) is SYS.
- The user has the EXEMPT ACCESS POLICY privilege.

Supported CTXCAT Preferences

index set *index_set*
Specifies the index set preference to create the CTXCAT index. Index set preferences name the columns that make up your subindexes. Any column that is named in an index set column list cannot have a NULL value in any row of the base table, or else you get an error.

Always ensure that your columns have non-null values before and after indexing.

See "Creating a CTXCAT Index" on page 1-54.

Index Performance and Size Considerations

Although a CTXCAT index offers query performance benefits, creating this type of index has its costs. The time that it takes Oracle Text to create a CTXCAT index depends on the total size of the index.

The total size of a CTXCAT index is directly related to:

- Total text to be indexed
- Number of component indexes in the index set
- Number of columns in the base table that make up the component indexes

Having many component indexes in your index set also degrades DML performance because more indexes must be updated.

Because of these added costs in creating a CTXCAT index, you should carefully consider the query performance benefit that each component index gives your application before adding it to your index set.

> **See Also:** *Oracle Text Application Developer's Guide* for more information about creating CTXCAT indexes and the benefits

Other CTXCAT Preferences

When you create an index of type CTXCAT, you can use the supported index preferences listed in Table 1–6 in the parameters string.

Table 1–6 Supported CTXCAT Index Preferences

Preference Class	Supported Types
Datastore	This preference class is not supported for CTXCAT.
Filter	This preference class is not supported for CTXCAT.
Lexer	BASIC_LEXER (index_themes attribute not supported) CHINESE_LEXER CHINESE_VGRAM_LEXER JAPANESE_LEXER JAPANESE_VGRAM_LEXER KOREAN_MORPH_LEXER
Wordlist	BASIC_WORDLIST
Storage	BASIC_STORAGE
Stoplist	Supports single language stoplists only (BASIC_STOPLIST type).
Section Group	Only Field Section is supported for CTXCAT.

Unsupported Preferences and Parameters

When you create a CTXCAT index, you cannot specify datastore and filter preferences. For section group preferences, only the field section preference is supported. You also cannot specify language, format, or charset columns as with a CONTEXT index.

Creating a CTXCAT Index

This section gives a brief example for creating a CTXCAT index. For a more complete example, see *Oracle Text Application Developer's Guide*.

Consider a table called AUCTION with the following schema:

```
create table auction(
item_id number,
title varchar2(100),
category_id number,
price number,
bid_close date);
```

Assume that queries on the table involve a mandatory text query clause and optional structured conditions on price. Results must be sorted based on bid_close. This means that an index to support good response time for the structured and sorting criteria is required.

You can create a catalog index to support the different types of structured queries a user might enter. For structured queries, a CTXCAT index improves query performance over a context index.

To create the indexes, first, create the index set preference, next, optionally, add the storage preference, and, finally, add the required indexes to it:

```
begin
ctx_ddl.create_index_set('auction_iset');
ctx_ddl.add_index('auction_iset','bid_close');
ctx_ddl.add_index('auction_iset','price, bid_close');
end;
```

Optionally, create the storage preference:

```
begin
 ctx_ddl.create_preference('auction_st_pref', 'BASIC_STORAGE');
 ctx_ddl.set_attribute('auction_st_pref', 'I_TABLE_CLAUSE',
                       'tablespace TEXT storage (initial 5M)');
 ctx_ddl.set_attribute('auction_st_pref', 'I_ROWID_INDEX_CLAUSE',
                        'tablespace TEXT storage (initial 5M)');
 ctx_ddl.set_attribute('auction_st_pref', 'I_INDEX_CLAUSE',
                        'tablespace TEXT storage (initial 5M) compress 2');
end;
/
```

Then, create the CTXCAT index with the CREATE INDEX statement as follows:

```
create index auction_titlex on AUCTION(title) indextype is CTXSYS.CTXCAT
parameters ('index set auction_iset storage auction_st_pref');
```

Querying a CTXCAT Index

To query the title column for the word *pokemon*, enter regular and mixed queries as follows:

```
select * from AUCTION where CATSEARCH(title, 'pokemon',NULL)> 0;
select * from AUCTION where CATSEARCH(title, 'pokemon', 'price < 50 order by
```

```
bid_close desc')> 0;
```

See Also: *Oracle Text Application Developer's Guide* for a complete CTXCAT example

Syntax for CTXRULE Index Type

The CTXRULE type is an index on a column containing a set of queries. Query this index with the MATCHES operator in the WHERE clause of a SELECT statement.

```
CREATE INDEX [schema.]index on [schema.]table(rule_col) INDEXTYPE IS
ctxsys.ctxrule
[PARAMETERS ('[lexer lexer_pref] [storage storage_pref]
[section group section_pref] [wordlist wordlist_pref]
[classifier classifier_pref]');
[PARALLEL n];
```

[*schema*.]*table*(*column*)
Specifies the name of the table and rule column to index. The rules can be query compatible strings, query template strings, or binary support vector machine rules.

The column you specify when you create a CTXRULE index must be VARCHAR2, CLOB or BLOB. No other types are supported for the CTXRULE type.

Attempting to create an index on a Virtual Private Database (VPD) protected table will fail unless one of the following is true:

- The VPD policy does not have the INDEX statement type turned on (which is the default).
- The policy function returns a null predicate for the current user.
- The user (index owner) is SYS.
- The user has the EXEMPT ACCESS POLICY privilege.

lexer_pref
Specifies the lexer preference to be used for processing queries and later for the documents to be classified with the MATCHES function.

With both classifiers SVN_CLASSFIER and RULE_CLASSIFIER, you can use the BASIC_LEXER, CHINESE_LEXER, JAPANESE_LEXER, or KOREAN_MORPH_LEXER lexer. (See "Classifier Types" on page 2-68 and "Lexer Types" on page 2-30.)

For processing queries, these lexers support the following operators: ABOUT, STEM, AND, NEAR, NOT, OR, and WITHIN.

The thesaural operators (BT*, NT*, PT, RT, SYN, TR, TRSYS, TT, and so on) are supported. However, these operators are expanded using a snapshot of the thesaurus at index time, not when the MATCHES function is entered. This means that if you change your thesaurus after you index, you must re-index your query set.

storage_pref
Specify the storage preference for the index on the queries.Use the storage preference to specify how the index tables are stored. See "Storage Types" on page 2-64.

section group
Specify the section group. This parameter does not affect the queries. It applies to sections in the documents to be classified. The following section groups are supported for the CTXRULE index type:

- BASIC_SECTION_GROUP

- HTML_SECTION_GROUP
- XML_SECTION_GROUP
- AUTO_SECTION_GROUP

See "Section Group Types" on page 2-66.

CTXRULE does not support special sections. It also does not support NDATA sections.

wordlist_pref
Specifies the wordlist preferences. This is used to enable stemming operations on query terms. See Wordlist Type on page 2-57.

classifier_pref
Specifies the classifier preference. See "Classifier Types" on page 2-68. You must use the same preference name you specify with CTX_CLS.TRAIN.

Example for Creating a CTXRULE Index

See *Oracle Text Application Developer's Guide* for a complete example of using the CTXRULE index type in a document routing application.

Syntax for CTXXPATH Index Type

This indextype if provided only for backward compatibility. Create a CTXXPATH index when you need to speed up existsNode() queries on an XMLType column.

```
CREATE INDEX [schema.]index on [schema.]table(XMLType column) INDEXTYPE IS
ctxsys.CTXXPATH
[PARAMETERS ('[storage storage_pref]
              [memory memsize]')];
```

[*schema*.]*table*(*column*)
Specifies the name of the table and column to index.

The column you specify when you create a CTXXPATH index must be XMLType. No other types are supported for the CTXXPATH index.

storage_pref
Specifies the storage preference for the index on the queries.

Use the storage preference to specify how the index tables are stored. See "Storage Types" on page 2-64 in Chapter 2, "Oracle Text Indexing Elements".

memory *memsize*
Specifies the amount of run-time memory to use for indexing. The syntax for memsize is as follows:

```
memsize = number[M|G|K]
```

M stands for megabytes, G stands for gigabytes, and K stands for kilobytes.

The value you specify for memsize must be between 1M and the value of MAX_INDEX_MEMORY in the CTX_PARAMETERS view. To specify a memory size larger than the MAX_INDEX_MEMORY, you must reset this parameter with CTX_ADM.SET_PARAMETER to be larger than or equal to memsize.

The default is the value specified for DEFAULT_INDEX_MEMORY in CTX_PARAMETERS.

CTXXPATH Examples

Index creation on an XMLType column:

```
CREATE INDEX xml_index ON xml_tab(col_xml) indextype is ctxsys.CTXXPATH;
```

Or

```
CREATE INDEX xml_index ON xml_tab(col_xml) indextype is ctxsys.CTXXPATH
 PARAMETERS('storage my_storage memory 40M');
```

Querying the table with `existsNode`:

```
select xml_id from xml_tab x where
x.col_xml.existsnode('/book/chapter[@title="XML"]') > 0;
```

> **See Also:** *Oracle XML DB Developer's Guide* for information about using the `CTXXPATH` index type

Related Topics

CTX_DDL.CREATE_PREFERENCE

CTX_DDL.CREATE_STOPLIST

CTX_DDL.CREATE_SECTION_GROUP

"ALTER INDEX" on page 1-2

"CATSEARCH" on page 1-21

DROP INDEX

> **Note:** This section describes the `DROP INDEX` statement as it pertains to dropping a Text domain index.
>
> For a complete description of the `DROP INDEX` statement, see *Oracle Database SQL Language Reference.*

Purpose

Use `DROP INDEX` to drop a specified Text index.

Syntax

```
DROP INDEX [schema.]index [force];
```

[force]
Optionally forces the index to be dropped. Use the `force` option when Oracle Text cannot determine the state of the index, such as when an indexing operation fails.

Oracle recommends against using this option by default. Use it only when a regular call to `DROP INDEX` fails.

Example

The following example drops an index named `doc_index` in the current user's database schema:

```
DROP INDEX doc_index;
```

Related Topics

"ALTER INDEX" on page 1-2

"CREATE INDEX" on page 1-36

MATCHES

Use the MATCHES operator to find all rows in a query table that match a given document. The document must be a plain text, HTML, or XML document.

The MATCHES operator also supports database links. You can identify a remote table or materialized view by appending @dblink to the end of its name. The dblink must be a complete or partial name for a database link to the database containing the remote table or materialized view. (Querying of remote views is not supported.)

This operator requires a CTXRULE index on your set of queries.

When the SVM_CLASSIFIER classifier type is used, MATCHES returns a score in the range 0 to 100; a higher number indicates a greater confidence in the match. Use the label parameter and MATCH_SCORE to obtain this number. Then use the matching score to apply a category-specific threshold to a particular category.

If the SVM_CLASSIFIER type is not used, then this operator returns either 100 (the document matches the criteria) or 0 (the document does not match).

Limitation

If the optimizer chooses to use the functional query invocation with a MATCHES query, your query will fail.

Syntax

```
MATCHES(
[schema.]column,
document VARCHAR2 or CLOB
[,label INTEGER])
RETURN NUMBER;
```

column
Specifies the column containing the indexed query set.

document
Specifies the document to be classified. The document can be plain text, HTML, or XML. Binary formats are not supported.

label
Optionally specifies the label that identifies the score generated by the MATCHES operator. Use this label with MATCH_SCORE.

Matches Example

The following example creates a table querytable, and populates it with classification names and associated rules. It then creates a CTXRULE index.

The example enters the MATCHES query with a document string to be classified. The SELECT statement returns all rows (queries) that are satisfied by the document:

```
create table querytable (classification varchar2(64), text varchar2(4000));
insert into querytable values ('common names', 'smith OR jones OR brown');
insert into querytable values ('countries', 'United States OR Great Britain OR
France');
insert into querytable values ('Oracle DB', 'oracle NEAR database');

create index query_rule on querytable(text) indextype is ctxsys.ctxrule;
```

```
SELECT classification FROM querytable WHERE MATCHES(text, 'Smith is a common name
in the United States') > 0;


CLASSIFICATION
----------------------------------------------------------------
common names
countries
```

Related Topics

"MATCH_SCORE" on page 1-61

"Syntax for CTXRULE Index Type" on page 1-55

CTX_CLS.TRAIN on page 6-2

Oracle Text Application Developer's Guide contains extended examples of simple and supervised classification, which make use of the `MATCHES` operator.

MATCH_SCORE

Use the MATCH_SCORE operator in a statement to return scores produced by a MATCHES query.

The MATCH_SCORE operator also supports database links. You can identify a remote table or materialized view by appending @dblink to the end of its name. The dblink must be a complete or partial name for a database link to the database containing the remote table or materialized view. (Querying of remote views is not supported.)

When the SVM_CLASSIFIER classifier type is used, this operator returns a score in the range 0 to 100. Use the matching score to apply a category-specific threshold to a particular category.

If the SVM_CLASSIFIER classifier is not used, then this operator returns either 100 (the document matches the criteria) or 0 (the document does not match).

Syntax

```
MATCH_SCORE(label NUMBER)
```

label
Specifies a number to identify the score produced by the query. Use this number to identify the MATCHES clause that returns this score.

Example

To get the matching score, use:

```
select cat_id, match_score(1) from training_result where matches(profile,
text,1)>0;
```

Related Topics

"MATCHES" on page 1-59

SCORE

Use the SCORE operator in a SELECT statement to return the score values produced by a CONTAINS query. The SCORE operator can be used in a SELECT, ORDER BY, or GROUP BY clause.

The SCORE operator also supports database links. You can identify a remote table or materialized view by appending @dblink to the end of its name. The dblink must be a complete or partial name for a database link to the database containing the remote table or materialized view. (Querying of remote views is not supported.)

Syntax

```
SCORE(label NUMBER)
```

label
Specifies a number to identify the score produced by the query. Use this number to identify the CONTAINS clause that returns this score.

Example

Single CONTAINS

When the SCORE operator is called (for example, in a SELECT clause), the CONTAINS clause must reference the score label value as in the following example:

```
SELECT SCORE(1), title from newsindex
          WHERE CONTAINS(text, 'oracle', 1) > 0 ORDER BY SCORE(1) DESC;
```

Multiple CONTAINS

Assume that a news database stores and indexes the title and body of news articles separately. The following query returns all the documents that include the words *Oracle* in their title and *java* in their body. The articles are sorted by the scores for the first CONTAINS (*Oracle*) and then by the scores for the second CONTAINS (*java*).

```
SELECT title, body, SCORE(10), SCORE(20)
FROM news
WHERE CONTAINS (news.title, 'Oracle', 10) > 0 OR
CONTAINS (news.body, 'java', 20) > 0
ORDER BY SCORE(10), SCORE(20);
```

Related Topics

"CONTAINS" on page 1-28

Appendix F, "The Oracle Text Scoring Algorithm"

2

Oracle Text Indexing Elements

Oracle Text provides indexing elements for creating Oracle Text indexes and for specifying indexing preferences. This chapter describes the indexing elements that you can use to create an Oracle Text index.

The following topics are discussed in this chapter:

Overview

When you use the CREATE INDEX statement to create an index or the ALTER INDEX statement to manage an index, you can optionally specify indexing preferences, stoplists, and section groups in the parameter string.

Specifying a preference, stoplist, or section group answers one of the following questions about the way Oracle Text indexes text:

Preference Class	Answers the Question
Datastore	How are your documents stored?
Filter	How can the documents be converted to plain text?
Lexer	What language is being indexed?
Wordlist	How should stem and fuzzy queries be expanded?
Storage	How should the index tables be stored?

Preference Class	Answers the Question
Stop List	What words or themes are not to be indexed?
Section Group	Is querying within sections enabled, and how are the document sections defined?

This chapter describes how to set each preference. Enable an option by creating a preference with one of the types described in this chapter.

For example, to specify that your documents are stored in external files, you can create a datastore preference called `mydatastore` using the FILE_DATASTORE type. Specify `mydatastore` as the datastore preference in the parameter clause of the `CREATE INDEX` statement.

Creating Preferences

To create a datastore, lexer, filter, classifier, wordlist, or storage preference, use the CTX_DDL.CREATE_PREFERENCEprocedure and specify one of the types described in this chapter. For some types, you can also set attributes with the CTX_DDL.SET_ATTRIBUTE procedure.

An indexing *type* names a class of indexing objects that you can use to create an index *preference*. A type, therefore, is an abstract ID, while a preference is an entity that corresponds to a type. Many system-defined preferences have the same name as types (for example, `BASIC_LEXER`), but exact correspondence is not guaranteed. For example, the `DEFAULT_DATASTORE` preference uses the `DIRECT_DATASTORE` type, and there is no system preference corresponding to the `CHARSET_FILTER` type. Be careful in assuming the existence or nature of either indexing types or system preferences.

You specify indexing preferences with the `CREATE INDEX` and `ALTER INDEX` statements. Indexing preferences determine how your index is created. For example, lexer preferences indicate the language of the text to be indexed. You can create and specify your own user-defined preferences, or you can use system-defined preferences.

To create a stoplist, use the CTX_DDL.CREATE_STOPLIST procedure. Add stopwords to a stoplist with `CTX_DDL.ADD_STOPWORD`.

To create section groups, use CTX_DDL.CREATE_SECTION_GROUP and specify a section group type. Add sections to section groups with the `CTX_DDL.ADD_ZONE_SECTION` or `CTX_DDL.ADD_FIELD_SECTION` procedures.

Datastore Types

Use the datastore types to specify how your text is stored. To create a datastore preference, you must use one of the datastore types described in Table 2–1.

Table 2–1 Datastore Types

Datastore Type	Use When
DIRECT_DATASTORE	Data is stored internally in the text column. Each row is indexed as a single document.
MULTI_COLUMN_DATASTORE	Data is stored in a text table in more than one column. Columns are concatenated to create a virtual document, one for each row.

Table 2–1 (Cont.) Datastore Types

Datastore Type	Use When
DETAIL_DATASTORE	Data is stored internally in the text column. Document consists of one or more rows stored in a text column in a detail table, with header information stored in a master table.
FILE_DATASTORE	Data is stored externally in operating system files. File names are stored in the text column, one for each row.
NESTED_DATASTORE	Data is stored in a nested table.
URL_DATASTORE	Data is stored externally in files located on an intranet or the Internet. Uniform Resource Locators (URLs) are stored in the text column.
USER_DATASTORE	Documents are synthesized at index time by a user-defined stored procedure.

DIRECT_DATASTORE

Use the DIRECT_DATASTORE type for text stored directly in the text column, one document for each row. The DIRECT_DATASTORE type has no attributes.

The following column types are supported: CHAR, VARCHAR, VARCHAR2, BLOB, CLOB, BFILE, XMLType, and URIType.

> **Note:** If your column is a BFILE, then the index owner must have *read* permission on all directories used by the BFILEs.

DIRECT_DATASTORE CLOB Example

The following example creates a table with a CLOB column to store text data. It then populates two rows with text data and indexes the table using the system-defined preference CTXSYS.DEFAULT_DATASTORE.

```
create table mytable(id number primary key, docs clob);

insert into mytable values(111555,'this text will be indexed');
insert into mytable values(111556,'this is a direct_datastore example');
commit;

create index myindex on mytable(docs)
  indextype is ctxsys.context
  parameters ('DATASTORE CTXSYS.DEFAULT_DATASTORE');
```

MULTI_COLUMN_DATASTORE

Use the MULTI_COLUMN_DATASTORE datastore when your text is stored in more than one column. During indexing, the system concatenates the text columns, tags the column text, and indexes the text as a single document. The XML-like tagging is optional. You can also set the system to filter and concatenate binary columns.

The data store MULTI_COLUMN_DATASTORE has the attributes shown in Table 2–2.

Table 2–2 MULTI_COLUMN_DATASTORE Attributes

Attribute	Attribute Value
columns	Specify a comma-delimited list of columns to be concatenated during indexing. You can also specify any allowed expression for the SELECT statement column list for the base table. This includes expressions, PL/SQL functions, column aliases, and so on. The NUMBER and DATE column types are supported. They are converted to text before indexing using the default format mask. The TO_CHAR function can be used in the column list for formatting. The RAW and BLOB columns are directly concatenated as binary data. The LONG, LONG RAW, NCHAR, and NCLOB data types, nested table columns, and collections are not supported. The column list is limited to 500 bytes.
filter	Specify a comma-delimited list of Y/N flags. Each flag corresponds to a column in the COLUMNS list and denotes whether to filter the column using the AUTO_FILTER. Specify one of the following allowed values: Y: Column is to be filtered with AUTO_FILTER N or no value: Column is not to be filtered (default)
delimiter	Specify the delimiter that separates column text as follows: COLUMN_NAME_TAG: Column text is set off by XML-like open and close tags (default). NEWLINE: Column text is separated with a newline.

Indexing and DML

To index, you must create a dummy column to specify in the CREATE INDEX statement. This column's contents are not made part of the virtual document, unless its name is specified in the columns attribute.

The index is synchronized only when the dummy column is updated. You can create triggers to propagate changes if needed.

MULTI_COLUMN_DATASTORE Restriction

You cannot create a multicolumn datastore with XMLType columns. MULTI_COLUMN_DATA_STORE does not support XMLType. You can create a CONTEXT index with an XMLType column, as described in Chapter 1, "Oracle Text SQL Statements and Operators".

MULTI_COLUMN_DATASTORE Example

The following example creates a multicolumn datastore preference called my_multi with three text columns:

```
begin
ctx_ddl.create_preference('my_multi', 'MULTI_COLUMN_DATASTORE');
ctx_ddl.set_attribute('my_multi', 'columns', 'column1, column2, column3');
end;
```

MULTI_COLUMN_DATASTORE Filter Example

The following example creates a multicolumn datastore preference and denotes that the bar column is to be filtered with the AUTO_FILTER.

```
ctx_ddl.create_preference('MY_MULTI','MULTI_COLUMN_DATASTORE');
```

```
ctx_ddl.set_attribute('MY_MULTI', 'COLUMNS','foo,bar');
ctx_ddl.set_attribute('MY_MULTI','FILTER','N,Y');
```

The multicolumn datastore fetches the content of the `foo` and `bar` columns, filters `bar`, then composes the compound document as:

```
<FOO>
foo contents
</FOO>
<BAR>
bar filtered contents (probably originally HTML)
</BAR>
```

The N flags do not need not be specified, and there does not need to be a flag for every column. Only the Y flags must be need to be specified, with commas to denote to which column they apply. For instance:

```
ctx_ddl.create_preference('MY_MULTI','MULTI_COLUMN_DATASTORE');
ctx_ddl.set_attribute('MY_MULTI', 'COLUMNS','foo,bar,zoo,jar');
ctx_ddl.set_attribute('MY_MULTI','FILTER',',,Y');
```

This filters only the column `zoo`.

Tagging Behavior

During indexing, the system creates a virtual document for each row. The virtual document is composed of the contents of the columns concatenated in the listing order with column name tags automatically added. For example:

```
create table mc(id number primary key, name varchar2(10), address varchar2(80));
insert into mc values(1, 'John Smith', '123 Main Street');

exec ctx_ddl.create_preference('mymds', 'MULTI_COLUMN_DATASTORE');
exec ctx_ddl.set_attibute('mymds', 'columns', 'name, address');
```

This produces the following virtual text for indexing:

```
<NAME>
John Smith
</NAME>
<ADDRESS>
123 Main Street
</ADDRESS>
```

The system indexes the text between the tags, ignoring the tags themselves.

Indexing Columns as Sections

To index the tags as sections, you can optionally create field sections with `BASIC_SECTION_GROUP`.

> **Note:** No section group is created when you use the `MULTI_COLUMN_DATASTORE`. To create sections for these tags, you must create a section group.

When you use expressions or functions, the tag is composed of the first 30 characters of the expression unless a column alias is used.

For example, if your expression is as follows:

```
exec ctx_ddl.set_attibute('mymds', 'columns', '4 + 17');
```

then it produces the following virtual text:

```
<4 + 17>
21
</4 + 17>
```

If your expression is as follows:

```
exec ctx_ddl.set_attibute('mymds', 'columns', '4 + 17 col1');
```

then it produces the following virtual text:

```
<col1>
21
<col1>
```

The tags are in uppercase unless the column name or column alias is in lowercase and surrounded by double quotation marks. For example:

```
exec ctx_ddl.set_attibute('mymds', 'COLUMNS', 'foo');
```

This produces the following virtual text:

```
<FOO>
content of foo
</FOO>
```

For lowercase tags, use the following:

```
exec ctx_ddl.set_attibute('mymds', 'COLUMNS', 'foo "foo"');
```

This expression produces:

```
<foo>
content of foo
</foo>
```

DETAIL_DATASTORE

Use the DETAIL_DATASTORE type for text stored directly in the database in detail tables, with the indexed text column located in the master table.

The DETAIL_DATASTORE type has the attributes described in Table 2–3.

Table 2–3 DETAIL_DATASTORE Attributes

Attribute	Attribute Value
binary	Specify TRUE for Oracle Text to add *no* newline character after each detail row. Specify FALSE for Oracle Text to add a newline character (\n) after each detail row automatically.
detail_table	Specify the name of the detail table (OWNER.TABLE if necessary).
detail_key	Specify the name of the detail table foreign key column.
detail_lineno	Specify the name of the detail table sequence column.
detail_text	Specify the name of the detail table text column.

Synchronizing Master/Detail Indexes

Changes to the detail table do not trigger re-indexing when you synchronize the index. Only changes to the indexed column in the master table triggers a re-index when you synchronize the index.

You can create triggers on the detail table to propagate changes to the indexed column in the master table row.

Example Master/Detail Tables

This example illustrates how master and detail tables are related to each other.

Master Table Example Master tables define the documents in a master/detail relationship. Assign an identifying number to each document. The following table is an example master table, called `my_master`:

Column Name	Column Type	Description
`article_id`	`NUMBER`	Document ID, unique for each document (primary key)
`author`	`VARCHAR2(30)`	Author of document
`title`	`VARCHAR2(50)`	Title of document
`body`	`CHAR(1)`	Dummy column to specify in `CREATE INDEX`

> **Note:** Your master table must include a primary key column when you use the `DETAIL_DATASTORE` type.

Detail Table Example Detail tables contain the text for a document, whose content is usually stored across a number of rows. The following detail table `my_detail` is related to the master table `my_master` with the `article_id` column. This column identifies the master document to which each detail row (sub-document) belongs.

Column Name	Column Type	Description
`article_id`	`NUMBER`	Document ID that relates to master table
`seq`	`NUMBER`	Sequence of document in the master document defined by `article_id`
`text`	`VARCHAR2`	Document text

Detail Table Example Attributes In this example, the `DETAIL_DATASTORE` attributes have the following values:

Attribute	Attribute Value
`binary`	`TRUE`
`detail_table`	`my_detail`
`detail_key`	`article_id`
`detail_lineno`	`seq`
`detail_text`	`text`

Use CTX_DDL.CREATE_PREFERENCE to create a preference with DETAIL_DATASTORE. Use CTX_DDL.SET_ATTRIBUTE to set the attributes for this preference as described earlier. The following example shows how this is done:

```
begin
ctx_ddl.create_preference('my_detail_pref', 'DETAIL_DATASTORE');
ctx_ddl.set_attribute('my_detail_pref', 'binary', 'true');
ctx_ddl.set_attribute('my_detail_pref', 'detail_table', 'my_detail');
ctx_ddl.set_attribute('my_detail_pref', 'detail_key', 'article_id');
ctx_ddl.set_attribute('my_detail_pref', 'detail_lineno', 'seq');
ctx_ddl.set_attribute('my_detail_pref', 'detail_text', 'text');
end;
```

Master/Detail Index Example To index the document defined in this master/detail relationship, specify a column in the master table using the CREATE INDEX statement. The column you specify must be one of the allowed types.

This example uses the body column, whose function is to enable the creation of the master/detail index and to improve readability of the code. The my_detail_pref preference is set to DETAIL_DATASTORE with the required attributes:

```
CREATE INDEX myindex on my_master(body) indextype is ctxsys.context
parameters('datastore my_detail_pref');
```

In this example, you can also specify the title or author column to create the index. However, if you do so, changes to these columns will trigger a re-index operation.

FILE_DATASTORE

The FILE_DATASTORE type is used for text stored in files accessed through the local file system.

> **Note:** The FILE_DATASTORE type may not work with certain types of remote-mounted file systems.

The FILE_DATASTORE type has the attributes described Table 2–4.

Table 2–4 FILE_DATASTORE Attributes

Attribute	Attribute Value
path	*path1*:*path2*:*pathn*
filename_charset	*name*

path
Specifies the full directory path name of the files stored externally in a file system. When you specify the full directory path as such, you need to include only file names in your text column.

You can specify multiple paths for the path attribute, with each path separated by a colon (:) on UNIX and semicolon(;) on Windows. File names are stored in the text column in the text table.

If you do not specify a path for external files with this attribute, then Oracle Text requires that the path be included in the file names stored in the text column.

filename_charset
Specifies a valid Oracle character set name (maximum length 30 characters) to be used by the file datastore for converting file names. In general, the Oracle database can use a different character set than the operating system. This can lead to problems in finding files (which may raise DRG-11513 errors) when the indexed column contains characters that are not convertible to the operating system character set. By default, the file datastore will convert the file name to WE8ISO8859p1 for ASCII platforms or WE8EBCDIC1047 for EBCDIC platforms.

However, this may not be sufficient for applications with multibyte character sets for both the database and the operating system, because neither WE8ISO8859p1 nor WE8EBCDIC1047 supports multibyte characters. The attribute `filename_charset` rectifies this problem. If specified, then the datastore will convert from the database character set to the specified character set rather than to ISO8859 or EBCDIC.

If the `filename_charset` attribute is the same as the database character set, then the file name is used as is. If `filename_charset` is not a valid character set, then the error "DRG-10763: value %s is not a valid character set" is raised.

PATH Attribute Limitations

The `PATH` attribute has the following limitations:

- If you specify a `PATH` attribute, then you can only use a simple file name in the indexed column. You cannot combine the `PATH` attribute with a path as part of the file name. If the files exist in multiple folders or directories, you must leave the `PATH` attribute unset, and include the full file name, with `PATH`, in the indexed column.
- On Windows systems, the files must be located on a local drive. They cannot be on a remote drive, whether the remote drive is mapped to a local drive letter.

FILE_DATASTORE and Security

File and URL datastores enable access to files on the actual database disk. This may be undesirable when security is an issue since any user can browse the file system that is accessible to the Oracle user. The `FILE_ACCESS_ROLE` system parameter can be used to set the name of a database role that is authorized to create an index using `FILE` or `URL` datastores. If set, any user attempting to create an index using `FILE` or `URL` datastores must have this role, or the index creation will fail. Only `SYS` can set `FILE_ACCESS_ROLE`, and an error will be raised if any other user tries to modify it. If `FILE_ACCESS_ROLE` is left at the default of NULL, access is disallowed. Thus, by default, users are not able to create indexes that use the file or URL datastores. Users can, if desired, set `FILE_ACCESS_ROLE` to `PUBLIC` if they want to preserve the behavior from earlier releases.

For example, the following statement sets the name of the database role:

```
ctx_adm.set_parameter('FILE_ACCESS_ROLE','TOPCAT');
```

where TOPCAT is the role that is authorized to create an index on a file or URL datastore. The `CREATE INDEX` operation will fail when a user that does not have an authorized role tries to create an index on a file or URL datastore. For example:

```
CREATE INDEX myindex ON mydocument(TEXT) INDEXTYPE IS ctxsys.context
  PARAMETERS('DATASTORE ctxsys.file_datastore')
```

In this case, if the user does not have the role TOPCAT, then index creation will fail and return an error. For users who have the TOPCAT role, the index creation will proceed normally.

The authorized role name is checked any time the datastore is accessed. This includes index creation, index sync, and calls to document services, such as CTX_DOC.HIGHLIGHT.

FILE_DATASTORE Example

This example creates a file datastore preference called COMMON_DIR that has a path of /mydocs:

```
begin
 ctx_ddl.create_preference('COMMON_DIR','FILE_DATASTORE');
 ctx_ddl.set_attribute('COMMON_DIR','PATH','/mydocs');
end;
```

When you populate the table mytable, you need only insert file names. The path attribute tells the system where to look during the indexing operation.

```
create table mytable(id number primary key, docs varchar2(2000));
insert into mytable values(111555,'first.txt');
insert into mytable values(111556,'second.txt');
commit;
```

Create the index as follows:

```
create index myindex on mytable(docs)
  indextype is ctxsys.context
  parameters ('datastore COMMON_DIR');
```

URL_DATASTORE

Use the URL_DATASTORE type for text stored:

- In files on the World Wide Web (accessed through HTTP or FTP)
- In files in the local file system (accessed through the file protocol)

Store each URL in a single text field.

URL Syntax

The syntax of a URL you store in a text field is as follows (with brackets indicating optional parameters):

```
[URL:]<access_scheme>://<host_name>[:<port_number>]/[<url_path>]
```

The access_scheme string can be either *ftp*, *http*, or *file*. For example:

```
http://mymachine.us.oracle.com/home.html
```

> **Note:** The login:password@ syntax within the URL is supported only for the ftp access scheme.

Because this syntax is partially compliant with the RFC 1738 specification, the following restriction holds for the URL syntax: The URL must contain only printable ASCII characters. Non-printable ASCII characters and multibyte characters must be escaped with the %*xx* notation, where *xx* is the hexadecimal representation of the special character.

URL_DATASTORE Attributes

URL_DATASTORE has the following attributes:

Table 2–5 URL_DATASTORE Attributes

Attribute	Attribute Value
timeout	The value of this attribute is ignored. This is provided for backward compatibility.
maxthreads	The value of this attribute is ignored. URL_DATASTORE is single-threaded. This is provided for backward compatibility.
urlsize	The value of this attribute is ignored. This is provided for backward compatibility.
maxurls	The value of this attribute is ignored. This is provided for backward compatibility.
maxdocsize	The value of this attribute is ignored. This is provided for backward compatibility.
http_proxy	Specify the host name of http proxy server. Optionally specify port number with a colon in the form hostname:port.
ftp_proxy	Specify the host name of ftp proxy server. Optionally specify port number with a colon in the form hostname:port.
no_proxy	Specify the domain for no proxy server. Use a comma separated string of up to 16 domain names.

timeout

The value of this attribute is ignored. This is provided for backward compatibility.

maxthreads

The value of this attribute is ignored. URL_DATASTORE is single-threaded. This is provided for backward compatibility.

urlsize

The value of this attribute is ignored. This is provided for backward compatibility.

maxdocsize

The value of this attribute is ignored. This is provided for backward compatibility.

maxurls

The value of this attribute is ignored. This is provided for backward compatibility.

http_proxy

Specify the fully qualified name of the host machine that serves as the HTTP proxy (gateway) for the machine on which Oracle Text is installed. You can optionally specify port number with a colon in the form hostname:port.

You must set this attribute if the machine is in an intranet that requires authentication through a proxy server to access Web files located outside the firewall.

ftp_proxy

Specify the fully qualified name of the host machine that serves as the FTP proxy (gateway) for the server on which Oracle Text is installed. You can optionally specify a port number with a colon in the form hostname:port.

This attribute must be set if the machine is in an intranet that requires authentication through a proxy server to access Web files located outside the firewall.

no_proxy
Specify a string of domains (up to sixteen, separated by commas) that are found in most, if not all, of the machines in your intranet. When one of the domains is encountered in a host name, no request is sent to the server(s) specified for `ftp_proxy` and `http_proxy`. Instead, the request is processed directly by the host machine identified in the URL.

For example, if the string *us.example.com*, *uk.example.com* is entered for `no_proxy`, any URL requests to machines that contain either of these domains in their host names are not processed by your proxy server(s).

URL_DATASTORE and Security

For a discussion of how to control file access security for file and URL datastores, refer to "FILE_DATASTORE and Security" on page 2-9.

URL_DATASTORE Example

This example creates a `URL_DATASTORE` preference called `URL_PREF` for which the `http_proxy`, `no_proxy`, and `timeout` attributes are set. The defaults are used for the attributes that are not set.

```
begin
 ctx_ddl.create_preference('URL_PREF','URL_DATASTORE');
 ctx_ddl.set_attribute('URL_PREF','HTTP_PROXY','www-proxy.us.oracle.com');
 ctx_ddl.set_attribute('URL_PREF','NO_PROXY','us.oracle.com');
 ctx_ddl.set_attribute('URL_PREF','Timeout','300');
end;
```

Create the table and insert values into it:

```
create table urls(id number primary key, docs varchar2(2000));
insert into urls values(111555,'http://context.us.oracle.com');
insert into urls values(111556,'http://www.sun.com');
commit;
```

To create the index, specify `URL_PREF` as the datastore:

```
create index datastores_text on urls ( docs )
  indextype is ctxsys.context
  parameters ( 'Datastore URL_PREF' );
```

USER_DATASTORE

Use the `USER_DATASTORE` type to define stored procedures that synthesize documents during indexing. For example, a user procedure might synthesize author, date, and text columns into one document to have the author and date information be part of the indexed text.

`USER_DATASTORE` has the following attributes:

Table 2–6 USER_DATASTORE Attributes

Attribute	Attribute Value
`procedure`	Specify the procedure that synthesizes the document to be indexed. This procedure can be owned by any user and must be executable by the index owner.

Table 2–6 (Cont.) USER_DATASTORE Attributes

Attribute	Attribute Value
output_type	Specify the data type of the second argument to procedure. Valid values are CLOB, BLOB, CLOB_LOC, BLOB_LOC, or VARCHAR2. The default is CLOB. When you specify CLOB_LOC, BLOB_LOC, you indicate that no temporary CLOB or BLOB is needed, because your procedure copies a locator to the IN/OUT second parameter.

procedure
Specify the name of the procedure that synthesizes the document to be indexed. This specification must be in the form PROCEDURENAME or PACKAGENAME.PROCEDURENAME. You can also specify the schema owner name.

The procedure you specify must have two arguments defined as follows:

```
procedure (r IN ROWID, c IN OUT NOCOPY output_type)
```

The first argument r must be of type ROWID. The second argument c must be of type output_type. NOCOPY is a compiler hint that instructs Oracle Text to pass parameter c by reference if possible.

> **Note:** The procedure name and its arguments can be named anything. The arguments r and c are used in this example for simplicity.

The stored procedure is called once for each row indexed. Given the rowid of the current row, procedure must write the text of the document into its second argument, whose type you specify with output_type.

Constraints

The following constraints apply to procedure:

- It can be owned by any user, but the user must have database permissions to execute procedure correctly
- It must be executable by the index owner
- It must not enter DDL or transaction control statements, like COMMIT

Editing Procedure after Indexing

When you change or edit the stored procedure, indexes based on it will not be notified, so you must manually re-create such indexes. So if the stored procedure makes use of other columns, and those column values change, the row will not be re-indexed. The row is re-indexed only when the indexed column changes.

output_type
Specify the datatype of the second argument to procedure. You can use either CLOB, BLOB, CLOB_LOC, BLOB_LOC, or VARCHAR2.

USER_DATASTORE with CLOB Example

Consider a table in which the author, title, and text fields are separate, as in the articles table defined as follows:

```
create table articles(
    id       number,
```

```
    author   varchar2(80),
    title    varchar2(120),
    text     clob );
```

The author and title fields are to be part of the indexed document text. Assume user appowner writes a stored procedure with the user datastore interface that synthesizes a document from the text, author, and title fields:

```
create procedure myproc(rid in rowid, tlob in out clob nocopy) is
  begin
      for c1 in (select author, title, text from articles
                  where rowid = rid)
      loop
   dbms_lob.writeappend(tlob, length(c1.title), c1.title);
   dbms_lob.writeappend(tlob, length(c1.author), c1.author);
   dbms_lob.writeappend(tlob, length(c1.text), c1.text);
       end loop;
    end;
```

This procedure takes in a rowid and a temporary CLOB locator, and concatenates all the article's columns into the temporary CLOB. The for loop executes only once.

The user appowner creates the preference as follows:

```
begin
ctx_ddl.create_preference('myud', 'user_datastore');
ctx_ddl.set_attribute('myud', 'procedure', 'myproc');
ctx_ddl.set_attribute('myud', 'output_type', 'CLOB');
end;
```

When appowner creates the index on articles(text) using this preference, the indexing operation sees author and title in the document text.

USER_DATASTORE with BLOB_LOC Example

The following procedure might be used with OUTPUT_TYPE BLOB_LOC:

```
procedure myds(rid in rowid, dataout in out nocopy blob)
is
  l_dtype varchar2(10);
  l_pk    number;
begin
  select dtype, pk into l_dtype, l_pk from mytable where rowid = rid;
  if (l_dtype = 'MOVIE') then
    select movie_data into dataout from movietab where fk = l_pk;
  elsif (l_dtype = 'SOUND') then
    select sound_data into dataout from soundtab where fk = l_pk;
  end if;
end;
```

The user appowner creates the preference as follows:

```
begin
ctx_ddl.create_preference('myud', 'user_datastore');
ctx_ddl.set_attribute('myud', 'procedure', 'myproc');
ctx_ddl.set_attribute('myud', 'output_type', 'blob_loc');
end;
```

NESTED_DATASTORE

Use the nested datastore type to index documents stored as rows in a nested table.

Table 2–7 NESTED_DATASTORE Attributes

Attribute	Attribute Value
nested_column	Specify the name of the nested table column.This attribute is required. Specify only the column name. Do not specify schema owner or containing table name.
nested_type	Specify the type of nested table. This attribute is required. You must provide owner name and type.
nested_lineno	Specify the name of the attribute in the nested table that orders the lines. This is like DETAIL_LINENO in detail datastore. This attribute is required.
nested_text	Specify the name of the column in the nested table type that contains the text of the line. This is like DETAIL_TEXT in detail datastore. This attribute is required. LONG column types are not supported as nested table text columns.
binary	Specify FALSE for Oracle Text to automatically insert a newline character when synthesizing the document text. If you specify TRUE, Oracle Text does not do this. This attribute is not required. The default is FALSE.

When using the nested table datastore, you must index a dummy column, because the extensible indexing framework disallows indexing the nested table column. See the example.

DML on the nested table is not automatically propagated to the dummy column used for indexing. For DML on the nested table to be propagated to the dummy column, your application code or trigger must explicitly update the dummy column.

Filter defaults for the index are based on the type of the nested_text column.

During validation, Oracle Text checks that the type exists and that the attributes you specify for nested_lineno and nested_text exist in the nested table type. Oracle Text does not check that the named nested table column exists in the indexed table.

NESTED_DATASTORE Example

This section shows an example of using the NESTED_DATASTORE type to index documents stored as rows in a nested table.

Create the Nested Table The following code creates a nested table and a storage table mytab for the nested table:

```
create type nt_rec as object (
  lno number, -- line number
  ltxt varchar2(80) -- text of line
);

create type nt_tab as table of nt_rec;
create table mytab (
   id number primary key, -- primary key
   dummy char(1), -- dummy column for indexing
   doc nt_tab -- nested table
)
nested table doc store as myntab;
```

Insert Values into Nested Table The following code inserts values into the nested table for the parent row with ID equal to 1.

```
insert into mytab values (1, null, nt_tab());
```

```
insert into table(select doc from mytab where id=1) values (1, 'the dog');
insert into table(select doc from mytab where id=1) values (2, 'sat on mat ');
commit;
```

Create Nested Table Preferences The following code sets the preferences and attributes for the NESTED_DATASTORE according to the definitions of the nested table type nt_tab and the parent table mytab:

```
begin
-- create nested datastore pref
ctx_ddl.create_preference('ntds','nested_datastore');

-- nest tab column in main table
ctx_ddl.set_attribute('ntds','nested_column', 'doc');

-- nested table type
ctx_ddl.set_attribute('ntds','nested_type', 'scott.nt_tab');

-- lineno column in nested table
ctx_ddl.set_attribute('ntds','nested_lineno','lno');

--text column in nested table
ctx_ddl.set_attribute('ntds','nested_text', 'ltxt');
end;
```

Create Index on Nested Table The following code creates the index using the nested table datastore:

```
create index myidx on mytab(dummy) -- index dummy column, not nest table
indextype is ctxsys.context parameters ('datastore ntds');
```

Query Nested Datastore The following select statement queries the index built from a nested table:

```
select * from mytab where contains(dummy, 'dog and mat')>0;
-- returns document 1, because it has dog in line 1 and mat in line 2.
```

Filter Types

Use the filter types to create preferences that determine how text is filtered for indexing. Filters enable word processor documents, formatted documents, plain text, HTML, and XML documents to be indexed.

For formatted documents, Oracle Text stores documents in their native format and uses filters to build interim plain text or HTML versions of the documents. Oracle Text indexes the words derived from the plain text or HTML version of the formatted document. The TMP_DIR environment variable sets the directory path for storing temporary files created by the filter.

To create a filter preference, you must use one of the following types:

Table 2–8 Filter Types

Filter	When Used
CHARSET_FILTER	Character set converting filter.
AUTO_FILTER	Auto filter for filtering formatted documents.
NULL_FILTER	No filtering required. Use for indexing plain text, HTML, or XML documents.

Table 2–8 (Cont.) Filter Types

Filter	When Used
MAIL_FILTER	Use the MAIL_FILTER to transform RFC-822, RFC-2045 messages in to text that can be indexed.
USER_FILTER	User-defined external filter to be used for custom filtering.
PROCEDURE_FILTER	User-defined stored procedure filter to be used for custom filtering.

CHARSET_FILTER

Use the CHARSET_FILTER to convert documents from a non-database character set to the character set used by the database.

CHARSET_FILTER has the attribute described in Table 2–9.

Table 2–9 CHARSET_FILTER Attributes

Attribute	Attribute Value
charset	Specify the Globalization Support name of source character set. If you specify UTF16AUTO, then this filter automatically detects the if the character set is UTF16 big- or little-endian. Specify JAAUTO for Japanese character set auto-detection. This filter automatically detects the custom character specification in JA16EUC or JA16SJIS and converts to the database character set. This filter is useful in Japanese when your data files have mixed character sets. JAAUTO can only be specified on a database whose character set is JA16EUC, JA16SJIS, or UTF8. Specify AUTO to have CHARSET_FILTER automatically detect and convert character sets that Oracle Database supports, as shown in Table 2–10.

When the charset column or attribute is set to AUTO, the CHARSET_FILTER automatically detects the document character set and converts the document from the detected character set to the database character set. CHARSET_FILTER can detect the supported character sets shown in Table 2–10.

Table 2–10 Character Sets Supported for CHARSET_FILTER Auto-detection

Character Set	
AL16UTF16	JA16EUC
AL32UTF8	JA16SJIS
AR8ISO8859P6	KO16KSC5601
AR8MSWIN1256	TH8TISASCII
CL8ISO8859P5	WE8ISO8859P1
CL8KOI8R	WE8ISO8859P9
CL8MSWIN1251	WE8MSWIN1252
EE8ISO8859P2	ZHS16CGB231280
EE8MSWIN1250	ZHS32GB18030
EL8ISO8859P7	ZHT16BIG5
EL8MSWIN1253	WE8MSWIN1252

See Also: *Oracle Database Globalization Support Guide* for more information about the supported globalization character sets

UTF-16 Big- and Little-Endian Detection

If your character set is UTF-16, then you can specify UTF16AUTO to automatically detect big- or little-endian data. Oracle Text does so by examining the first two bytes of the document row.

If the first two bytes are 0xFE, 0xFF, the document is recognized as big-endian and the remainder of the document minus those two bytes is passed on for indexing.

If the first two bytes are 0xFF, 0xFE, the document is recognized as little-endian and the remainder of the document minus those two bytes is passed on for indexing.

If the first two bytes are anything else, the document is assumed to be big-endian and the whole document including the first two bytes is passed on for indexing.

Indexing Mixed-Character Set Columns

A mixed character set column is one that stores documents of different character sets. For example, a text table might store some documents in WE8ISO8859P1 and others in UTF8.

To index a table of documents in different character sets, you must create your base table with a character set column. In this column, specify the document character set on a per-row basis. To index the documents, Oracle Text converts the documents into the database character set.

Character set conversion works with the `CHARSET_FILTER`. When the charset column is `NULL` or not recognized, Oracle Text assumes the source character set is the one specified in the charset attribute.

Note: Character set conversion also works with the `AUTO_FILTER` when the document format column is set to `TEXT`.

Indexing Mixed-Character Set Example For example, create the table with a charset column:

```
create table hdocs (
     id number primary key,
     fmt varchar2(10),
     cset varchar2(20),
     text varchar2(80)
);
```

Create a preference for this filter:

```
begin
cxt_ddl.create_preference('cs_filter', 'CHARSET_FILTER');
ctx_ddl.set_attribute('cs_filter', 'charset', 'UTF8');
end;
/
```

Insert plain-text documents and name the character set:

```
insert into hdocs values(1, 'text', 'WE8ISO8859P1', '/docs/iso.txt');
insert into hdocs values (2, 'text', 'UTF8', '/docs/utf8.txt');
commit;
```

Create the index and name the charset column:

```
create index hdocsx on hdocs(text) indextype is ctxsys.context
  parameters ('datastore ctxsys.file_datastore
  filter cs_filter
  format column fmt
  charset column cset');
```

AUTO_FILTER

The AUTO_FILTER is a universal filter that filters most document formats, including PDF and Microsoft Word documents. Use it for indexing both single-format and mixed-format columns. This filter automatically bypasses plain text, HTML, XHTML, SGML, and XML documents.

> **See Also:** Appendix B, "Oracle Text Supported Document Formats", for a list of the formats supported by AUTO_FILTER, and to learn more about how to set up your environment

> **Note:** The AUTO_FILTER replaces the INSO_FILTER, which has been deprecated. While every effort has been made to ensure maximal backward compatibility between the two filters, so that applications using INSO_FILTER will continue to work without modification, some differences may arise. Users should therefore use AUTO_FILTER in their new programs and, when possible, replace instances of INSO_FILTER, and any system preferences or constants that make use of it, in older applications.

The AUTO_FILTER preference has the following attributes:

Table 2–11 AUTO_FILTER Attributes

Attribute	Attribute Value
timeout	Specify the AUTO_FILTER timeout in seconds. Use a number between 0 and 42,949,672. Default is 120. Setting this value to 0 disables the feature. How this wait period is used depends on how you set timeout_type. This feature is disabled for rows for which the corresponding charset and format column cause the AUTO_FILTER to bypass the row, such as when format is marked TEXT. Use this feature to prevent the Oracle Text indexing operation from waiting indefinitely on a hanging filter operation.
timeout_type	Specify either HEURISTIC or FIXED. Default is HEURISTIC. Specify HEURISTIC for Oracle Text to check every TIMEOUT seconds if output from Outside In HTML Export has increased. The operation terminates for the document if output has not increased. An error is recorded in the CTX_USER_INDEX_ERRORS view and Oracle Text moves to the next document row to be indexed. Specify FIXED to terminate the Outside In HTML Export processing after TIMEOUT seconds regardless of whether filtering was progressing normally or just hanging. This value is useful when indexing throughput is more important than taking the time to successfully filter large documents.

Table 2–11 (Cont.) AUTO_FILTER Attributes

Attribute	Attribute Value
output_formatting	Setting this attribute has no effect on filter performance or filter output. It is maintained for backward compatibility.

Indexing Formatted Documents

To index a text column containing formatted documents such as Microsoft Word, use the AUTO_FILTER. This filter automatically detects the document format. Use the CTXSYS.AUTO_FILTER system-defined preference in the parameter clause as follows:

```
create index hdocsx on hdocs(text) indextype is ctxsys.context
  parameters ('datastore ctxsys.file_datastore
  filter ctxsys.auto_filter');
```

> **Note:** The CTXSYS.AUTO_FILTER replaces CTXSYS.INSO_FILTER, which has been deprecated. Programs making use of CTXSYS.INSO_FILTER should still work. New programs should use CTXSYS.AUTO_FILTER.

Explicitly Bypassing Plain Text or HTML in Mixed Format Columns

A mixed-format column is a text column containing more than one document format, such as a column that contains Microsoft Word, PDF, plain text, and HTML documents.

The AUTO_FILTER can index mixed-format columns, automatically bypassing plain text, HTML, and XML documents. However, if you prefer not to depend on the built-in bypass mechanism, you can explicitly tag your rows as text and cause the AUTO_FILTER to ignore the row and not process the document in any way.

The format column in the base table enables you to specify the type of document contained in the text column. You can specify the following document types: TEXT, BINARY, and IGNORE. During indexing, the AUTO_FILTER ignores any document typed TEXT, assuming the charset column is not specified. (The difference between a document with a TEXT format column type and one with an IGNORE type is that the TEXT document is indexed, but ignored by the filter, while the IGNORE document is not indexed at all. Use IGNORE to overlook documents such as image files, or documents in a language that you do not want to index. IGNORE can be used with any filter type.)

To set up the AUTO_FILTER bypass mechanism, you must create a format column in your base table.

For example:

```
create table hdocs (
     id number primary key,
     fmt varchar2(10),
     text varchar2(80)
);
```

Assuming you are indexing mostly Word documents, you specify BINARY in the format column to filter the Word documents. Alternatively, to have the AUTO_FILTER ignore an HTML document, specify TEXT in the format column.

For example, the following statements add two documents to the text table, assigning one format as BINARY and the other TEXT:

```
insert into hdocs values(1, 'binary', '/docs/myword.doc');
```

```
insert in hdocs values (2, 'text', '/docs/index.html');
commit;
```

To create the index, use CREATE INDEX and specify the format column name in the parameter string:

```
create index hdocsx on hdocs(text) indextype is ctxsys.context
  parameters ('datastore ctxsys.file_datastore
  filter ctxsys.auto_filter
  format column fmt');
```

If you do not specify TEXT or BINARY for the format column, BINARY is used.

Note: You need not specify the format column in CREATE INDEX when using the AUTO_FILTER.

Character Set Conversion With AUTO_FILTER

The AUTO_FILTER converts documents to the database character set when the document format column is set to TEXT. In this case, the AUTO_FILTER looks at the charset column to determine the document character set.

If the charset column value is not an Oracle Text character set name, the document is passed through without any character set conversion.

Note: You need not specify the charset column when using the AUTO_FILTER.

If you do specify the charset column and do not specify the format column, the AUTO_FILTER works like the CHARSET_FILTER, except that in this case there is no Japanese character set auto-detection.

See Also: "CHARSET_FILTER" on page 2-17.

NULL_FILTER

Use the NULL_FILTER type when plain text or HTML is to be indexed and no filtering needs to be performed. NULL_FILTER has no attributes.

Indexing HTML Documents

If your document set is entirely HTML, Oracle recommends that you use the NULL_FILTER in your filter preference.

For example, to index an HTML document set, specify the system-defined preferences for NULL_FILTER and HTML_SECTION_GROUP as follows:

```
create index myindex on docs(htmlfile) indextype is ctxsys.context
  parameters('filter ctxsys.null_filter
  section group ctxsys.html_section_group');
```

See Also: For more information on section groups and indexing HTML documents, see "Section Group Types" on page 2-66.

MAIL_FILTER

Use MAIL_FILTER to transform RFC-822, RFC-2045 messages into indexable text. The following limitations apply to the input:

- Documents must be US-ASCII
- Lines must not be longer than 1024 bytes
- Documents must be syntactically valid with regard to RFC-822.

Behavior for invalid input is not defined. Some deviations may be robustly handled by the filter without error. Others may result in a fetch-time or filter-time error.

The MAIL_FILTER has the following attributes:

Table 2–12 MAIL_FILTER Attributes

Attribute	Attribute Value
INDEX_FIELDS	Specify a colon-separated list of fields to preserve in the output. These fields are transformed to tag markup. For example, if INDEX_FIELDS is set to "FROM": From: Scott Tiger becomes: <FROM>Scott Tiger</FROM> Only top-level fields are transformed in this way.
AUTO_FILTER_TIMEOUT	Specify a timeout value for the AUTO_FILTER filtering invoked by the mail filter. Default is 60. (Replaces the INSO_TIMEOUT attribute and is backward compatible with INSO_TIMEOUT.)
AUTO_FILTER_OUTPUT_FORMATTING	Specify either TRUE or FALSE. Default is TRUE. This attribute replaces the previous INSO_OUTPUT_FORMATTING attribute. However, it has no effect in the current release.
PART_FIELD_STYLE	Specify how fields occurring in lower-level parts and identified by the INDEX_FIELDS attribute should be transformed. The fields of the top-level message part identified by INDEX_FIELDS are always transformed to tag markup (see the previous description of INDEX_FIELDS); PART_FIELD_STYLE controls the transformation of subsequent parts; for example, attached e-mails. Possible values include IGNORE (the default), in which the part fields are not included for indexing; TAG, in which the part field names are transformed to tags, as occurs with top-level part fields; FIELD, in which the part field names are preserved as fields, not as tags; and TEXT, in which the part field names are eliminated and only the field content is preserved for indexing. See "Mail_Filter Example" on page 2-24 for an example of how PART_FIELD_STYLE works.

Filter Behavior

This filter behaves in the following way for each document:

- Read and remove header fields
- Decode message body if needed, depending on Content-transfer-encoding field
- Take action depending on the Content-Type field value and the user-specified behavior specified in a mail filter configuration file. (See "About the Mail Filter Configuration File" on page 2-23.) The possible actions are:
  - produce the body in the output text (INCLUDE). If no character set is encountered in the INCLUDE parts in the Content-Type header field, then Oracle defaults to the value specified in the character set column in the base

table. Name your populated character set column in the parameter string of the CREATE INDEX command.

  - AUTO_FILTER the body contents (AUTO_FILTER directive).
  - remove the body contents from the output text (IGNORE)
- If no behavior is specified for the type in the configuration file, then the defaults are as follows:
  - text/*: produce body in the output text
  - application/*: AUTO_FILTER the body contents
  - image/*, audio/*, video/*, model/*: ignore
- Multipart messages are parsed, and the mail filter applied recursively to each part. Each part is appended to the output.
- All text produced will be charset-converted to the database character set, if needed.

About the Mail Filter Configuration File

The MAIL_FILTER filter makes use of a mail filter configuration file, which contains directives specifying how a mail document should be filtered. The mail filter configuration file is a editable text file. Here you can override default behavior for each Content-Type. The configuration file also contains IANA-to-Oracle Globalization Support character set name mappings.

The location of the file must be in ORACLE_HOME/ctx/config. The name of the file to use is stored in the new system parameter MAIL_FILTER_CONFIG_FILE. On install, this is set to drmailfl.txt, which has useful default contents.

Oracle recommends that you create your own mail filter configuration files to avoid overwrite by the installation of a new version or patch set. The mail filter configuration file should be in the database character set.

Mail File Configuration File Structure The file has two sections, BEHAVIOR and CHARSETS. Indicate the start of the behavior section as follows:

```
[behavior]
```

Each line following starts with a mime type, then whitespace, then behavior specification. The MIME type can be a full TYPE/SUBTYPE or just TYPE, which will apply to all subtypes of that type. TYPE/SUBTYPE specification overrides TYPE specification, which overrides default behavior. Behavior can be INCLUDE, AUTO_FILTER, or IGNORE (see "Filter Behavior" on page 2-22 for definitions). For instance:

```
application/zip     IGNORE
application/msword  AUTO_FILTER
model               IGNORE
```

You cannot specify behavior for "multipart" or "message" types. If you do, such lines are ignored. Duplicate specification for a type replaces earlier specifications.

Comments can be included in the mail configuration file by starting lines with the # symbol.

The charset mapping section begins with

```
[charsets]
```

Lines consist of an IANA name, then whitespace, then an Oracle Globalization Support charset name, like:

```
US-ASCII     US7ASCI
ISO-8859-1   WE8ISO8859P1
```

This file is the only way the mail filter gets the mappings. There are no defaults.

When you change the configuration file, the changes affect only the documents indexed after that point. You must flush the shared pool after changing the file.

Mail_Filter Example

Suppose there is an e-mail with the following form, in which other e-mails with different subject lines are attached to this e-mail:

```
To:  somebody@someplace
Subject:  mainheader
Content-Type:  multipart/mixed
. . .
Content-Type: text/plain
X-Ref:  some_value
Subject:  subheader 1
. . .
Content-Type:  text/plain
X-Control:  blah blah blah
Subject:  subheader 2
. . .
```

Set INDEX_FIELDS to be "Subject" and, initially, PART_FIELD_STYLE to IGNORE.

```
CTX_DDL.CREATE_PREFERENCE('my_mail_filt', 'mail_filter');
CTX_DDL_SET_ATTRIBUTE(my_mail_filt', 'INDEX_FILES', 'subject');
CTX_DDL.SET ATTRIBUTE ('my_mail_filt', 'PART_FIELD_STYLE', 'ignore');
```

Now when the index is created, the file will be indexed as follows:

```
<SUBJECT>mainheader</SUBJECT>
```

If PART_FIELD_STYLE is instead set to TAG, this becomes:

```
<SUBJECT>mainheader</SUBJECT>
<SUBJECT>subheader1</SUBJECT>
<SUBJECT>subheader2</SUBJECT>
```

If PART_FIELD_STYLE is set to FIELD instead, this is the result:

```
<SUBJECT>mainheader<SUBJECT>
SUBJECT:subheader1
SUBJECT:subheader2
```

Finally, if PART_FIELD_STYLE is instead set to TEXT, then the result is:

```
<SUBJECT>mainheader</SUBJECT>
subheader1
subheader2
```

USER_FILTER

Use the USER_FILTER type to specify an external filter for filtering documents in a column. USER_FILTER has the following attribute:

Table 2–13 USER_FILTER Attribute

Attribute	Attribute Value
command	Specify the name of the filter executable.

CAUTION: The USER_FILTER type introduces the potential for security threats. A database user granted the CTXAPP role could potentially use USER_FILTER to load a malicious application. Therefore, the DBA must safeguard against any combination of input and output file parameters that would enable the named filter executable to compromise system security.

command
Specify the executable for the single external filter that is used to filter all text stored in a column. If more than one document format is stored in the column, then the external filter specified for command must recognize and handle all such formats.

The executable that you specify must exist in the $ORACLE_HOME/ctx/bin directory on UNIX, and in the %ORACLE_HOME%/ctx/bin directory on Windows.

You must create your user-filter command with two parameters:

- The first parameter is the name of the input file to be read.
- The second parameter is the name of the output file to be written to.

If all the document formats are supported by AUTO_FILTER, then use AUTO_FILTER instead of USER_FILTER, unless additional tasks besides filtering are required for the documents.

Using USER_FILTER with Charset and Format Columns

USER_FILTER bypasses documents that do not need to be filtered. Its behavior is sensitive to the values of the format and charset columns. In addition, USER_FILTER performs character set conversion according to the charset column values.

Explicitly Bypassing Plain Text or HTML in Mixed Format Columns

A mixed-format column is a text column containing more than one document format, such as a column that contains Microsoft Word, PDF, plain text, and HTML documents.

The USER_FILTER executable can index mixed-format columns, automatically bypassing textual documents. However, if you prefer not to depend on the built-in bypass mechanism, you can explicitly tag your rows as text and cause the USER_FILTER executable to ignore the row and not process the document in any way.

The format column in the base table enables you to specify the type of document contained in the text column. You can specify the following document types: TEXT, BINARY, and IGNORE. During indexing, the USER_FILTER executable ignores any document typed TEXT, assuming the charset column is not specified. (The difference between a document with a TEXT format column type and one with an IGNORE type is that the TEXT document is indexed, but ignored by the filter, while the IGNORE document is not indexed at all. Use IGNORE to overlook documents such as image files, or documents in a language that you do not want to index. IGNORE can be used with any filter type.

To set up the USER_FILTER bypass mechanism, you must create a format column in your base table. For example:

```
create table hdocs (
   id number primary key,
   fmt varchar2(10),
   text varchar2(80)
);
```

Assuming you are indexing mostly Word documents, you specify BINARY in the format column to filter the Word documents. Alternatively, to have the USER_FILTER executable ignore an HTML document, specify TEXT in the format column.

For example, the following statements add two documents to the text table, assigning one format as BINARY and the other TEXT:

```
insert into hdocs values(1, 'binary', '/docs/myword.doc');
insert into hdocs values(2, 'text', '/docs/index.html');
commit;
```

Assuming that this file is named upcase.pl, create the filter preference as follows:

```
ctx_ddl.create_preference
   (
    preference_name => 'USER_FILTER_PREF',
    object_name     => 'USER_FILTER'
    );

ctx_ddl.set_attribute ('USER_FILTER_PREF', 'COMMAND', 'upcase.pl');
```

To create the index, use CREATE INDEX and specify the format column name in the parameter string:

```
create index hdocsx on hdocs(text) indextype is ctxsys.context
   parameters ('datastore ctxsys.file_datastore
   filter 'USER_FILTER_PREF'
   format column fmt');
```

If you do not specify TEXT or BINARY for the format column, BINARY is used.

Character Set Conversion with USER_FILTER

The USER_FILTER executable converts documents to the database character set when the document format column is set to TEXT. In this case, the USER_FILTER executable looks at the charset column to determine the document character set.

If the charset column value is not an Oracle Text character set name, the document is passed through without any character set conversion.

If you do specify the charset column and do not specify the format column, the USER_FILTER executable works like the CHARSET_FILTER, except that in this case, there is no Japanese character set auto-detection. See "CHARSET_FILTER" on page 2-17 for more information regarding CHARSET_FILTER.

User Filter Example

The following example shows a Perl script to be used as the user filter. This script converts the input text file specified in the first argument to uppercase and writes the output to the location specified in the second argument.

```
#!/usr/local/bin/perl
```

```
open(IN, $ARGV[0]);
open(OUT, ">".$ARGV[1]);

while (<IN>)
{
  tr/a-z/A-Z/;
  print OUT;
}

close (IN);
close (OUT);
```

Assuming that this file is named `upcase.pl`, create the filter preference as follows:

```
begin
  ctx_ddl.create_preference
    (
      preference_name => 'USER_FILTER_PREF',
      object_name     => 'USER_FILTER'
    );
  ctx_ddl.set_attribute
    ('USER_FILTER_PREF','COMMAND','upcase.pl');
end;
```

Create the index in SQL*Plus as follows:

```
create index user_filter_idx on user_filter ( docs )
  indextype is ctxsys.context
  parameters ('FILTER USER_FILTER_PREF');
```

PROCEDURE_FILTER

Use the `PROCEDURE_FILTER` type to filter your documents with a stored procedure. The stored procedure is called each time a document needs to be filtered.

Table 2–14 lists the attributes for `PROCEDURE_FILTER`.

Table 2–14 PROCEDURE_FILTER Attributes

Attribute	Purpose	Allowable Values
`procedure`	Name of the filter stored procedure.	Any procedure. The procedure can be a PL/SQL stored procedure.
`input_type`	Type of input argument for stored procedure.	`VARCHAR2, BLOB, CLOB, FILE`
`output_type`	Type of output argument for stored procedure.	`VARCHAR2, CLOB, FILE`
`rowid_parameter`	Include rowid parameter?	`TRUE/FALSE`
`format_ parameter`	Include format parameter?	`TRUE/FALSE`
`charset_ parameter`	Include charset parameter?	`TRUE/FALSE`

procedure
Specify the name of the stored procedure to use for filtering. The procedure can be a PL/SQL stored procedure. The procedure can be a safe callout, or call a safe callout.

With the rowid_parameter, format_parameter, and charset_parameter set to FALSE, the procedure can have one of the following signatures:

```
PROCEDURE(IN BLOB, IN OUT NOCOPY CLOB)
PROCEDURE(IN CLOB, IN OUT NOCOPY CLOB)
PROCEDURE(IN VARCHAR, IN OUT NOCOPY CLOB)
PROCEDURE(IN BLOB, IN OUT NOCOPY VARCHAR2)
PROCEDURE(IN CLOB, IN OUT NOCOPY VARCHAR2)
PROCEDURE(IN VARCHAR2, IN OUT NOCOPY VARCHAR2)
PROCEDURE(IN BLOB, IN VARCHAR2)
PROCEDURE(IN CLOB, IN VARCHAR2)
PROCEDURE(IN VARCHAR2, IN VARCHAR2)
```

The first argument is the content of the unfiltered row, output by the datastore. The second argument is for the procedure to pass back the filtered document text.

The procedure attribute is mandatory and has no default.

input_type
Specify the type of the input argument of the filter procedure. You can specify one of the following types:

Type	Description
procedure	Name of the filter stored procedure.
input_type	Type of input argument for stored procedure.
output_type	Type of output argument for stored procedure.
rowid_parameter	Include rowid parameter?

The input_type attribute is not mandatory. If not specified, then BLOB is the default.

output_type
Specify the type of output argument of the filter procedure. You can specify one of the following types:

Type	Description
CLOB	The output argument is IN OUT NOCOPY CLOB. Your procedure must write the filtered content to the CLOB passed in.
VARCHAR2	The output argument is IN OUT NOCOPY VARCHAR2. Your procedure must write the filtered content to the VARCHAR2 variable passed in.
FILE	The output argument must be IN VARCHAR2. On entering the filter procedure, the output argument is the name of a temporary file. The filter procedure must write the filtered contents to this named file. Using a FILE output type is useful only when the procedure is a safe callout, which can write to the file.

The output_type attribute is not mandatory. If not specified, then CLOB is the default.

rowid_ parameter
When you specify TRUE, the rowid of the document to be filtered is passed as the first parameter, before the input and output parameters.

For example, with INPUT_TYPE BLOB, OUTPUT_TYPE CLOB, and ROWID_PARAMETER TRUE, the filter procedure must have the signature as follows:

```
procedure(in rowid, in blob, in out nocopy clob)
```

This attribute is useful for when your procedure requires data from other columns or tables. This attribute is not mandatory. The default is FALSE.

format_parameter
When you specify TRUE, the value of the format column of the document being filtered is passed to the filter procedure before input and output parameters, but after the rowid parameter, if enabled.

Specify the name of the format column at index time in the parameters string, using the keyword 'format column <columnname>'. The parameter type must be IN VARCHAR2.

The format column value can be read by means of the rowid parameter, but this attribute enables a single filter to work on multiple table structures, because the format attribute is abstracted and does not require the knowledge of the name of the table or format column.

FORMAT_PARAMETERis not mandatory. The default is FALSE.

charset_parameter
When you specify TRUE, the value of the charset column of the document being filtered is passed to the filter procedure before input and output parameters, but after the rowid and format parameter, if enabled.

Specify the name of the charset column at index time in the parameters string, using the keyword 'charset column <columnname>'. The parameter type must be IN VARCHAR2.

CHARSET_PARAMETERattribute is not mandatory. The default is FALSE.

Parameter Order

ROWID_PARAMETER, FORMAT_PARAMETER, and CHARSET_PARAMETERare all independent. The order is rowid, the format, then charset. However, the filter procedure is passed only the minimum parameters required.

For example, assume that INPUT_TYPE is BLOB and OUTPUT_TYPE is CLOB. If your filter procedure requires all parameters, then the procedure signature must be:

```
(id IN ROWID, format IN VARCHAR2, charset IN VARCHAR2, input IN BLOB, output IN
OUT NOCOPY CLOB)
```

If your procedure requires only the ROWID, then the procedure signature must be:

```
(id IN ROWID,input IN BLOB, ouput IN OUT NOCOPY CLOB)
```

Procedure Filter Execute Requirements

To create an index using a PROCEDURE_FILTER preference, the index owner must have *execute* permission on the procedure.

Error Handling

The filter procedure can raise any errors needed through the normal PL/SQL raise_application_error facility. These errors are propagated to the CTX_USER_INDEX_ERRORS view or reported to the user, depending on how the filter is invoked.

Procedure Filter Preference Example

Consider a filter procedure CTXSYS.NORMALIZE that you define with the following signature:

```
PROCEDURE NORMALIZE(id IN ROWID, charset IN VARCHAR2, input IN CLOB,
output IN OUT NOCOPY VARCHAR2);
```

To use this procedure as your filter, set up your filter preference as follows:

```
begin
ctx_ddl.create_preference('myfilt', 'procedure_filter');
ctx_ddl.set_attribute('myfilt', 'procedure', 'normalize');
ctx_ddl.set_attribute('myfilt', 'input_type', 'clob');
ctx_ddl.set_attribute('myfilt', 'output_type', 'varchar2');
ctx_ddl.set_attribute('myfilt', 'rowid_parameter', 'TRUE');
ctx_ddl.set_attribute('myfilt', 'charset_parameter', 'TRUE');
end;
```

Lexer Types

Use the lexer preference to specify the language of the text to be indexed. To create a lexer preference, you must use one of the lexer types described in Table 2–15.

Table 2–15 Lexer Types

Type	Description
BASIC_LEXER	Lexer for extracting tokens from text in languages, such as English and most western European languages that use white space delimited words.
MULTI_LEXER	Lexer for indexing tables containing documents of different languages such as English, German, and Japanese.
CHINESE_VGRAM_LEXER	Lexer for extracting tokens from Chinese text.
CHINESE_LEXER	Lexer for extracting tokens from Chinese text. This lexer offers benefits over the CHINESE_VGRAM lexer: ■ Generates a smaller index ■ Better query response time ■ Generates real world tokens resulting in better query precision ■ Supports stop words
JAPANESE_VGRAM_LEXER	Lexer for extracting tokens from Japanese text.
JAPANESE_LEXER	Lexer for extracting tokens from Japanese text. This lexer offers the following advantages over the JAPANESE_VGRAM lexer: ■ Generates smaller index ■ Better query response time ■ Generates real world tokens resulting in better precision
KOREAN_MORPH_LEXER	Lexer for extracting tokens from Korean text.
USER_LEXER	Lexer you create to index a particular language.
WORLD_LEXER	Lexer for indexing tables containing documents of different languages; autodetects languages in a document.

BASIC_LEXER

Use the BASIC_LEXER type to identify tokens for creating Text indexes for English and all other supported whitespace-delimited languages.

The BASIC_LEXER also enables base-letter conversion, composite word indexing, case-sensitive indexing and alternate spelling for whitespace-delimited languages that have extended character sets.

In English and French, you can use the BASIC_LEXER to enable theme indexing.

> **Note:** Any processing that the lexer does to tokens before indexing (for example, removal of characters, and base-letter conversion) are also performed on query terms at query time. This ensures that the query terms match the form of the tokens in the Text index.

BASIC_LEXER supports any database character set.

BASIC_LEXER has the attributes shown in Table 2–16.

Table 2–16 BASIC_LEXER Attributes

Attribute	Attribute Value
continuation	*characters*
numgroup	*characters*
numjoin	*characters*
printjoins	*characters*
punctuations	*characters*
skipjoins	*characters*
startjoins	*non alphanumeric characters that occur at the beginning of a token* (string)
endjoins	*non alphanumeric characters that occur at the end of a token* (string)
whitespace	*characters* (string)
newline	*NEWLINE (\n)*
	CARRIAGE_RETURN (\r)
base_letter	NO (disabled)
	YES (enabled)
base_letter_type	GENERIC (default)
	SPECIFIC
override_base_letter	TRUE
	FALSE (default)
mixed_case	NO (disabled)
	YES (enabled)
composite	DEFAULT (no composite word indexing, default)
	GERMAN (German composite word indexing)
	DUTCH (Dutch composite word indexing)

Table 2–16 (Cont.) BASIC_LEXER Attributes

Attribute	Attribute Value
index_stems	0 NONE
	1 ENGLISH
	2 DERIVATIONAL
	3 DUTCH
	4 FRENCH
	5 GERMAN
	6 ITALIAN
	7 SPANISH
index_themes	YES (enabled)
	NO (disabled, default)
	NO (disabled, default)
index_text	YES (enabled, default
	NO (disabled)
prove_themes	YES (enabled, default)
	NO (disabled)
theme_language	AUTO (default)
	(any Globalization Support language)
alternate_spelling	GERMAN (German alternate spelling)
	DANISH (Danish alternate spelling)
	SWEDISH (Swedish alternate spelling)
	NONE (No alternate spelling, default)
new_german_spelling	YES
	NO (default)

continuation
Specify the characters that indicate a word continues on the next line and should be indexed as a single token. The most common continuation characters are hyphen '-' and backslash '\'.

numgroup
Specify a single character that, when it appears in a string of digits, indicates that the digits are groupings within a larger single unit.

For example, comma ',' might be defined as a `numgroup` character because it often indicates a grouping of thousands when it appears in a string of digits.

numjoin
Specify the characters that, when they appear in a string of digits, cause Oracle Text to index the string of digits as a single unit or word.

For example, period '.' can be defined as numjoin characters because it often serves as decimal points when it appears in a string of digits.

Note: The default values for `numjoin` and `numgroup` are determined by the globalization support initialization parameters that are specified for the database.

In general, a value need not be specified for either `numjoin` or `numgroup` when creating a lexer preference for `BASIC_LEXER`.

printjoins
*S*pecify the non alphanumeric characters that, when they appear anywhere in a word (beginning, middle, or end), are processed as alphanumeric and included with the token in the Text index. This includes `printjoins` that occur consecutively.

For example, if the hyphen '-' and underscore '_' characters are defined as `printjoins`, terms such as *pseudo-intellectual* and *_file_* are stored in the Text index as *pseudo-intellectual* and *_file_*.

Note: If a `printjoins` character is also defined as a `punctuations` character, the character is only processed as an alphanumeric character if the character immediately following it is a standard alphanumeric character or has been defined as a `printjoins` or `skipjoins` character.

punctuations
Specify a list of non-alphanumeric characters that, when they appear at the end of a word, indicate the end of a sentence. The defaults are period '.', question mark '?', and exclamation point '!'.

Characters that are defined as `punctuations` are removed from a token before text indexing. However, if a `punctuations` character is also defined as a `printjoins` character, then the character is removed only when it is the last character in the token.

For example, if the period (.) is defined as both a `printjoins` and a `punctuations` character, then the following transformations take place during indexing and querying as well:

Token	Indexed Token
.doc	.doc
dog.doc	dog.doc
dog..doc	dog..doc
dog.	dog
dog...	dog..

In addition, `BASIC_LEXER` use `punctuations` characters in conjunction with `newline` and `whitespace` characters to determine sentence and paragraph delimiters for sentence/paragraph searching.

skipjoins
Specify the non-alphanumeric characters that, when they appear within a word, identify the word as a single token; however, the characters are not stored with the token in the Text index.

For example, if the hyphen character '-' is defined as a `skipjoins`, then the word *pseudo-intellectual* is stored in the Text index as *pseudointellectual*.

> **Note:** `Printjoins` and `skipjoins` are mutually exclusive. The same characters cannot be specified for both attributes.

startjoins/endjoins
For `startjoins`, specify the characters that when encountered as the first character in a token explicitly identify the start of the token. The character, as well as any other `startjoins` characters that immediately follow it, is included in the Text index entry for the token. In addition, the first `startjoins` character in a string of `startjoins` characters implicitly ends the previous token.

For `endjoins`, specify the characters that when encountered as the last character in a token explicitly identify the end of the token. The character, as well as any other `startjoins` characters that immediately follow it, is included in the Text index entry for the token.

The following rules apply to both `startjoins` and `endjoins`:

- The characters specified for `startjoins/endjoins` cannot occur in any of the other attributes for `BASIC_LEXER`.
- `startjoins/endjoins` characters can occur only at the beginning or end of tokens

Printjoins differ from endjoins and startjoins in that position does not matter. For example, *$35* will be indexed as one token if *$* is a `startjoin` or a `printjoin`, but as two tokens if it is defined as an `endjoin`.

whitespace
*S*pecify the characters that are treated as blank spaces between tokens. `BASIC_LEXER` uses `whitespace` characters in conjunction with `punctuations` and `newline` characters to identify character strings that serve as sentence delimiters for sentence and paragraph searching.

The predefined default values for `whitespace` are `space` and `tab`. These values cannot be changed. Specifying characters as `whitespace` characters adds to these defaults.

newline
*S*pecify the characters that indicate the end of a line of text. `BASIC_LEXER` uses `newline` characters in conjunction with punctuations and whitespace characters to identify character strings that serve as paragraph delimiters for sentence and paragraph searching.

The only valid values for `newline` are `NEWLINE` and `CARRIAGE_RETURN` (for carriage returns). The default is `NEWLINE`.

base_letter
Specify whether characters that have diacritical marks (umlauts, cedillas, acute accents, and so on) are converted to their base form before being stored in the Text index. The default is NO (base-letter conversion disabled). For more information on base-letter conversions and `base_letter_type`, see Base-Letter Conversion on page 15-2.

base_letter_type
Specify `GENERIC` or `SPECIFIC`.

The GENERIC value is the default and means that base letter transformation uses one transformation table that applies to all languages. For more information on base-letter conversions and base_letter_type, see "Base-Letter Conversion" on page 15-2.

override_base_letter
When base_letter is enabled at the same time as alternate_spelling, it is sometimes necessary to override base_letter to prevent unexpected results from serial transformations. See "Overriding Base-Letter Transformations with Alternate Spelling" on page 15-3. Default is FALSE.

mixed_case
Specify whether the lexer leaves the tokens exactly as they appear in the text or converts the tokens to all uppercase. The default is NO (tokens are converted to all uppercase).

> **Note:** Oracle Text ensures that word queries match the case sensitivity of the index being queried. As a result, if you enable case sensitivity for your Text index, queries against the index are always case sensitive.

composite
Specify whether composite word indexing is disabled or enabled for either GERMAN or DUTCH text. The default is DEFAULT (composite word indexing disabled).

Words that are usually one entry in a German dictionary are not split into composite stems, while words that aren't dictionary entries are split into composite stems.

To retrieve the indexed composite stems, you must enter a stem query, such as *$bahnhof*. The language of the wordlist stemmer must match the language of the composite stems.

Stemming User-Dictionaries

You can create a user-dictionary for your own language to customize how words are decomposed. These dictionaries are shown in Table 2–17.

Table 2–17 Stemming User-Dictionaries

Dictionary	Stemmer
$ORACLE_HOME/ctx/data/frlx/drfr.dct	French
$ORACLE_HOME/ctx/data/delx/drde.dct	German
$ORACLE_HOME/ctx/data/nllx/drnl.dct	Dutch
$ORACLE_HOME/ctx/data/itlx/drit.dct	Italian
$ORACLE_HOME/ctx/data/eslx/dres.dct	Spanish
$ORACLE_HOME/ctx/data/enlx/dren.dct	English and Derivational

Stemming user-dictionaries are not supported for languages other than those listed in Table 2–17.

The format for the user dictionary is as follows:

```
output term <tab> input term
```

The individual parts of the decomposed word must be separated by the # character. The following example entries are for the German word *Hauptbahnhof*:

```
Hauptbahnhof<tab>Haupt#Bahnhof
Hauptbahnhofes<tab>Haupt#Bahnhof
Hauptbahnhof<tab>Haupt#Bahnhof
Hauptbahnhoefe<tab>Haupt#Bahnhof
```

index_themes
Specify YES to index theme information in English or French. This makes ABOUT queries more precise. The index_themes and index_text attributes cannot both be NO. The default is YES.

You can set this parameter to TRUE for any index type, including CTXCAT. To enter an ABOUT query with CATSEARCH, use the query template with CONTEXT grammar.

Note: index_themes requires an installed knowledge base. A knowledge base may or may not have been installed with Oracle Text. For more information on knowledge bases, see *Oracle Text Application Developer's Guide*.

---

prove_themes
Specify YES to prove themes. Theme proving attempts to find related themes in a document. When no related themes are found, parent themes are eliminated from the document.

While theme proving is acceptable for large documents, short text descriptions with a few words rarely prove parent themes, resulting in poor recall performance with ABOUT queries.

Theme proving results in higher precision and less recall (less rows returned) for ABOUT queries. For higher recall in ABOUT queries and possibly less precision, you can disable theme proving. Default is YES.

The prove_themes attribute is supported for CONTEXT and CTXRULE indexes.

theme_language
Specify which knowledge base to use for theme generation when index_themes is set to YES. When index_themes is NO, setting this parameter has no effect on anything.

Specify any globalization support language or AUTO. You must have a knowledge base for the language you specify. This release provides a knowledge base in only English and French. In other languages, you can create your own knowledge base.

> **See Also:** "Adding a Language-Specific Knowledge Base" on page 14-7 in Chapter 14, "Oracle Text Utilities".

The default is AUTO, which instructs the system to set this parameter according to the language of the environment.

index_stems
Specify the stemmer to use for stem indexing. Choose one of the following stemmers:

NONE	GERMAN
DERIVATIONAL	ITALIAN
DUTCH	NORWEGIAN
ENGLISH	SPANISH
FRENCH	SWEDISH

Tokens are stemmed to a single base form at index time in addition to the normal forms. Indexing stems enables better query performance for stem ($) queries, such as *$computed*.

index_text
Specify `YES` to index word information. The index_themes and index_text attributes cannot both be `NO`.

The default is `NO`.

alternate_spelling
Specify either `GERMAN`, `DANISH`, or `SWEDISH` to enable the alternate spelling in one of these languages. Enabling alternate spelling enables you to query a word in any of its alternate forms.

Alternate spelling is off by default; however, in the language-specific scripts that Oracle provides in `admin/defaults` (`drdefd.sql` for German, `drdefdk.sql` for Danish, and `drdefs.sql` for Swedish), alternate spelling is turned on. If your installation uses these scripts, then alternate spelling is on. However, you can specify `NONE` for no alternate spelling. For more information about the alternate spelling conventions Oracle Text uses, see Alternate Spelling on page 15-2.

new_german_spelling
Specify whether the queries using the `BASIC_LEXER` return both traditional and reformed (new) spellings of German words. If `new_german_spelling` is set to `YES`, then both traditional and new forms of words are indexed. If it is set to `NO`, then the word will be indexed only as it as provided in the query. The default is `NO`.

> **See Also:** "New German Spelling" on page 15-2

BASIC_LEXER Example

The following example sets printjoin characters and disables theme indexing with the `BASIC_LEXER`:

```
begin
ctx_ddl.create_preference('mylex', 'BASIC_LEXER');
ctx_ddl.set_attribute('mylex', 'printjoins', '_-');
ctx_ddl.set_attribute ( 'mylex', 'index_themes', 'NO');
ctx_ddl.set_attribute ( 'mylex', 'index_text', 'YES');
end;
```

To create the index with no theme indexing and with printjoin characters set as described, enter the following statement:

```
create index myindex on mytable ( docs )
  indextype is ctxsys.context
  parameters ( 'LEXER mylex' );
```

MULTI_LEXER

Use MULTI_LEXER to index text columns that contain documents of different languages. For example, use this lexer to index a text column that stores English, German, and Japanese documents.

This lexer has no attributes.

You must have a language column in your base table. To index multi-language tables, specify the language column when you create the index.

Create a multi-lexer preference with CTX_DDL.CREATE_PREFERENCE. Add language-specific lexers to the multi-lexer preference with the CTX_DDL.ADD_SUB_LEXER procedure.

During indexing, the MULTI_LEXER examines each row's language column value and switches in the language-specific lexer to process the document.

The WORLD_LEXER lexer also performs multi-language indexing, but without the need for separate language columns (that is, it has automatic language detection). For more on WORLD_LEXER, see "WORLD_LEXER" on page 2-56.

Multi-language Stoplists

When you use the MULTI_LEXER, you can also use a multi-language stoplist for indexing.

> **See Also:** "Multi-Language Stoplists" on page 2-72.

MULTI_LEXER Example

Create the multi-language table with a primary key, a text column, and a language column as follows:

```
create table globaldoc (
   doc_id number primary key,
   lang varchar2(3),
   text clob
);
```

Assume that the table holds mostly English documents, with the occasional German or Japanese document. To handle the three languages, you must create three sub-lexers, one for English, one for German, and one for Japanese:

```
ctx_ddl.create_preference('english_lexer','basic_lexer');
ctx_ddl.set_attribute('english_lexer','index_themes','yes');
ctx_ddl.set_attribute('english_lexer','theme_language','english');

ctx_ddl.create_preference('german_lexer','basic_lexer');
ctx_ddl.set_attribute('german_lexer','composite','german');
ctx_ddl.set_attribute('german_lexer','mixed_case','yes');
ctx_ddl.set_attribute('german_lexer','alternate_spelling','german');

ctx_ddl.create_preference('japanese_lexer','japanese_vgram_lexer');
```

Create the multi-lexer preference:

```
ctx_ddl.create_preference('global_lexer', 'multi_lexer');
```

Because the stored documents are mostly English, make the English lexer the default using CTX_DDL.ADD_SUB_LEXER:

```
ctx_ddl.add_sub_lexer('global_lexer','default','english_lexer');
```

Now add the German and Japanese lexers in their respective languages with CTX_DDL.ADD_SUB_LEXER procedure. Also assume that the language column is expressed in the standard ISO 639-2 language codes, so add those as alternative values.

```
ctx_ddl.add_sub_lexer('global_lexer','german','german_lexer','ger');
ctx_ddl.add_sub_lexer('global_lexer','japanese','japanese_lexer','jpn');
```

Now create the index `globalx`, specifying the multi-lexer preference and the language column in the parameter clause as follows:

```
create index globalx on globaldoc(text) indextype is ctxsys.context
parameters ('lexer global_lexer language column lang');
```

Querying Multi-Language Tables

At query time, the multi-lexer examines the language setting and uses the sub-lexer preference for that language to parse the query.

If the language is not set, then the default lexer is used. Otherwise, the query is parsed and run as usual. The index contains tokens from multiple languages, so such a query can return documents in several languages. To limit your query to a given language, use a structured clause on the language column.

CHINESE_VGRAM_LEXER

The `CHINESE_VGRAM_LEXER` type identifies tokens in Chinese text for creating Text indexes.

CHINESE_VGRAM_LEXER Attribute

The `CHINESE_VGRAM_LEXER` has the following attribute:

Table 2–18 CHINESE_VGRAM_LEXER Attributes

Attribute	Attribute Value
`mixed_case_ASCII7`	Enable mixed-case (upper- and lower-case) searches of ASCII7 text (for example, *cat* and *Cat*). Allowable values are `YES` and `NO` (default).

Character Sets

You can use this lexer if your database uses one of the following character sets:

- AL32UTF8
- ZHS16CGB231280
- ZHS16GBK
- ZHS32GB18030
- ZHT32EUC
- ZHT16BIG5
- ZHT32TRIS
- ZHT16HKSCS
- ZHT16MSWIN950
- UTF8

CHINESE_LEXER

The CHINESE_LEXER type identifies tokens in traditional and simplified Chinese text for creating Oracle Text indexes.

This lexer offers the following benefits over the CHINESE_VGRAM_LEXER:

- generates a smaller index
- better query response time
- generates real word tokens resulting in better query precision
- supports stop words

Because the CHINESE_LEXER uses a different algorithm to generate tokens, indexing time is longer than with CHINESE_VGRAM_LEXER.

You can use this lexer if your database character is one of the Chinese or Unicode character sets supported by Oracle.

CHINESE_LEXER Attribute

The CHINESE_LEXER has the following attribute:

Table 2–19 CHINESE_LEXER Attributes

Attribute	Attribute Value
mixed_case_ASCII7	Enable mixed-case (upper- and lower-case) searches of ASCII7 text (for example, *cat* and *Cat*). Allowable values are YES and NO (default).

Customizing the Chinese Lexicon

You can modify the existing lexicon (dictionary) used by the Chinese lexer, or create your own Chinese lexicon, with the ctxlc command.

See Also: "Lexical Compiler (ctxlc)" on page 14-8 in Chapter 14, "Oracle Text Utilities"

JAPANESE_VGRAM_LEXER

The JAPANESE_VGRAM_LEXER type identifies tokens in Japanese for creating Text indexes. This lexer supports the stem ($) operator.

JAPANESE_VGRAM_LEXER Attributes

This lexer has the following attributes:

Table 2–20 JAPANESE_VGRAM_LEXER Attributes

Attribute	Attribute Value
delimiter	Specify whether to consider certain Japanese blank characters, such as a full-width forward slash or a full-width middle dot. ALL considers these characters, while NONE ignores them. Default is NONE.
mixed_case_ASCII7	Enable mixed-case (upper- and lower-case) searches of ASCII7 text (for example, *cat* and *Cat*). Allowable values are YES and NO (default).

JAPANESE_VGRAM_LEXER Character Sets

You can use this lexer if your database uses one of the following character sets:

- JA16SJIS
- JA16EUC
- UTF8
- AL32UTF8
- JA16EUCTILDE
- JA16EUCYEN
- JA16SJISTILDE
- JA16SJISYEN

JAPANESE_LEXER

The JAPANESE_LEXER type identifies tokens in Japanese for creating Text indexes. This lexer supports the stem ($) operator.

This lexer offers the following benefits over the JAPANESE_VGRAM_LEXER:

- generates a smaller index
- better query response time
- generates real word tokens resulting in better query precision

Because the JAPANESE_LEXER uses a new algorithm to generate tokens, indexing time is longer than with JAPANESE_VGRAM_LEXER.

Customizing the Japanese Lexicon

You can modify the existing lexicon (dictionary) used by the Japanese lexer, or create your own Japanese lexicon, with the ctxlc command.

> **See Also:** "Lexical Compiler (ctxlc)" on page 14-8 in Chapter 14, "Oracle Text Utilities"

JAPANESE_LEXER Attributes

This lexer has the following attributes:

Table 2–21 JAPANESE_LEXER Attributes

Attribute	Attribute Value
delimiter	Specify NONE or ALL to ignore certain Japanese blank characters, such as a full-width forward slash or a full-width middle dot. Default is NONE.
mixed_case_ASCII7	Enable mixed-case (upper- and lower-case) searches of ASCII7 text (for example, *cat* and *Cat*). Allowable values are YES and NO (default).

JAPANESE LEXER Character Sets

The JAPANESE_LEXER supports the following character sets:

- JA16SJIS
- JA16EUC

- UTF8
- AL32UTF8
- JA16EUCTILDE
- JA16EUCYEN
- JA16SJISTILDE
- JA16SJISYEN

Japanese Lexer Example

When you specify JAPANESE_LEXER for creating text index, the JAPANESE_LEXER resolves a sentence into words.

For example, the following compound word (*natural language institute*)

is indexed as three tokens:

To resolve a sentence into words, the internal dictionary is referenced. When a word cannot be found in the internal dictionary, Oracle Text uses the JAPANESE_VGRAM_LEXER to resolve it.

KOREAN_MORPH_LEXER

The KOREAN_MORPH_LEXER type identifies tokens in Korean text for creating Oracle Text indexes.

Supplied Dictionaries

The KOREAN_MORPH_LEXER uses four dictionaries:

Table 2–22 KOREAN_MORPH_LEXER Dictionaries

Dictionary	File
System	$ORACLE_HOME/ctx/data/kolx/drk2sdic.dat
Grammar	$ORACLE_HOME/ctx/data/kolx/drk2gram.dat
Stopword	$ORACLE_HOME/ctx/data/kolx/drk2xdic.dat
User-defined	$ORACLE_HOME/ctx/data/kolx/drk2udic.dat

The grammar, user-defined, and stopword dictionaries should be written using the KSC 5601 or MSWIN949 character sets. You can modify these dictionaries using the defined rules. The system dictionary must not be modified.

You can add unregistered words to the user-defined dictionary file. The rules for specifying new words are in the file.

Supported Character Sets

You can use KOREAN_MORPH_LEXER if your database uses one of the following character sets:

- KO16KSC5601
- KO16MSWIN949
- UTF8
- AL32UTF8

The KOREAN_MORPH_LEXER enables mixed-case searches.

Unicode Support

The KOREAN_MORPH_LEXER supports:

- Words in non-KSC5601 Korean characters defined in Unicode
- Supplementary characters

> **See Also:** For information on supplementary characters, see the *Oracle Database Globalization Support Guide*

Some Korean documents may have non-KSC5601 characters in them. As the KOREAN_MORPH_LEXER can recognize all possible 11,172 Korean (Hangul) characters, such documents can also be interpreted by using the UTF8 or AL32UTF8 character sets.

Use the AL32UTF8 character set for your database to extract surrogate characters. By default, the KOREAN_MORPH_LEXER extracts all series of surrogate characters in a document as one token for each series.

Limitations on Korean Unicode Support For conversion Hanja to Hangul (Korean), the KOREAN_MORPH_LEXER supports only the 4888 Hanja characters defined in KSC5601.

KOREAN_MORPH_LEXER Attributes

When you use the KOREAN_MORPH_LEXER, you can specify the following attributes:

Table 2–23 KOREAN_MORPH_LEXER Attributes

Attribute	Attribute Value
verb_adjective	Specify TRUE or FALSE to index verbs, adjectives, and adverbs. Default is FALSE.
one_char_word	Specify TRUE or FALSE to index one syllable. Default is FALSE.
number	Specify TRUE or FALSE to index number. Default is FALSE.
user_dic	Specify TRUE or FALSE to index user dictionary. Default is TRUE.
stop_dic	Specify TRUE of FALSE to use stop-word dictionary. Default is TRUE. The stop-word dictionary belongs to KOREAN_MORPH_LEXER.
composite	Specify indexing style of composite noun. Specify COMPOSITE_ONLY to index only composite nouns. Specify NGRAM to index all noun components of a composite noun. Specify COMPONENT_WORD to index single noun components of composite nouns as well as the composite noun itself. Default is COMPONENT_WORD. The following example describes the difference between NGRAM and COMPONENT_WORD.

Table 2–23 (Cont.) KOREAN_MORPH_LEXER Attributes

Attribute	Attribute Value
morpheme	Specify TRUE or FALSE for morphological analysis. If set to FALSE, tokens are created from the words that are divided by delimiters such as white space in the document. Default is TRUE.
to_upper	Specify TRUE or FALSE to convert English to uppercase. Default is TRUE.
hanja	Specify TRUE to index hanja characters. If set to FALSE, hanja characters are converted to hangul characters. Default is FALSE.
long_word	Specify TRUE to index long words that have more than 16 syllables in Korean. Default is FALSE.
japanese	Specify TRUE to index Japanese characters in Unicode (only in the 2-byte area). Default is FALSE.
english	Specify TRUE to index alphanumeric strings. Default is TRUE.

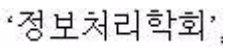
Limitations

Sentence and paragraph sections are not supported with the KOREAN_MORPH_LEXER.

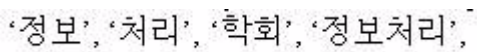
KOREAN_MORPH_LEXER Example: Setting Composite Attribute

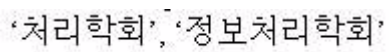
Use the composite attribute to control how composite nouns are indexed.

NGRAM Example When you specify NGRAM for the composite attribute, composite nouns are indexed with all possible component tokens. For example, the following composite noun (*information processing institute*)



is indexed as six tokens:





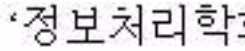
Specify NGRAM indexing as follows:

```
begin
ctx_ddl.create_preference('my_lexer','KOREAN_MORPH_LEXER');
ctx_ddl.set_attribute('my_lexer','COMPOSITE','NGRAM');
end
```

To create the index:

```
create index koreanx on korean(text) indextype is ctxsys.context
parameters ('lexer my_lexer');
```

COMPONENT_WORD Example When you specify COMPONENT_WORD for the composite attribute, composite nouns and their components are indexed. For example, the following composite noun (*information processing institute*)



is indexed as four tokens:

'정보처리학회',

'정보', '처리', '학회'

Specify COMPONENT_WORD indexing as follows:

```
begin
ctx_ddl.create_preference('my_lexer','KOREAN_MORPH_LEXER');
ctx_ddl.set_attribute('my_lexer','COMPOSITE','COMPONENT_WORD');
end
```

To create the index:

```
create index koreanx on korean(text) indextype is ctxsys.context
parameters ('lexer my_lexer');
```

USER_LEXER

Use USER_LEXER to plug in your own language-specific lexing solution. This enables you to define lexers for languages that are not supported by Oracle Text. It also enables you to define a new lexer for a language that is supported but whose lexer is inappropriate for your application.

The user-defined lexer you register with Oracle Text is composed of two routines that you must supply:

Table 2–24 User-Defined Routines for USER_LEXER

User-Defined Routine	Description
Indexing Procedure	Stored procedure (PL/SQL) which implements the tokenization of documents and stop words. Output must be an XML document as specified in this section.
Query Procedure	Stored procedure (PL/SQL) which implements the tokenization of query words. Output must be an XML document as specified in this section.

Limitations

The following features are not supported with the USER_LEXER:

- CTX_DOC.GIST and CTX_DOC.THEMES
- CTX_QUERY.HFEEDBACK
- ABOUT query operator
- CTXRULE index type
- VGRAM indexing algorithm

USER_LEXER Attributes

USER_LEXER has the following attributes:

Table 2–25 USER_LEXER Attributes

Attribute	Attribute Value
INDEX_PROCEDURE	Name of a stored procedure. No default provided.
INPUT_TYPE	VARCHAR2, CLOB. Default is CLOB.

Table 2–25 (Cont.) USER_LEXER Attributes

Attribute	Attribute Value
QUERY_PROCEDURE	Name of a stored procedure. No default provided.

INDEX_PROCEDURE

This callback stored procedure is called by Oracle Text as needed to tokenize a document or a stop word found in the stoplist object.

Requirements This procedure can be a PL/SQL stored procedure.

The index owner must have EXECUTE privilege on this stored procedure.

This stored procedure must not be replaced or dropped after the index is created. You can replace or drop this stored procedure after the index is dropped.

Parameters Two different interfaces are supported for the user-defined lexer indexing procedure:

- VARCHAR2 Interface
- CLOB Interface

Restrictions This procedure must not perform any of the following operations:

- Rollback
- Explicitly or implicitly commit the current transaction
- Enter any other transaction control statement
- Alter the session language or territory

The child elements of the root element tokens of the XML document returned must be in the same order as the tokens occur in the document or stop word being tokenized.

The behavior of this stored procedure must be deterministic with respect to all parameters.

INPUT_TYPE

Two different interfaces are supported for the User-defined lexer indexing procedure. One interface enables the document or stop word and the corresponding tokens encoded as XML to be passed as VARCHAR2 datatype whereas the other interface uses the CLOB datatype. This attribute indicates the interface implemented by the stored procedure specified by the INDEX_PROCEDURE attribute.

VARCHAR2 Interface BASIC_WORDLIST Attributes Table 2–33 describes the interface that enables the document or stop word from stoplist object to be tokenized to be passed as VARCHAR2 from Oracle Text to the stored procedure and for the tokens to be passed as VARCHAR2 as well from the stored procedure back to Oracle Text.

Your user-defined lexer indexing procedure should use this interface when all documents in the column to be indexed are smaller than or equal to 32512 bytes and the tokens can be represented by less than or equal to 32512 bytes. In this case the CLOB interface given in Table 2–27 can also be used, although the VARCHAR2 interface will generally perform faster than the CLOB interface.

This procedure must be defined with the following parameters:

Table 2–26 VARCHAR2 Interface for INDEX_PROCEDURES

Parameter Position	Parameter Mode	Parameter Datatype	Description
1	IN	VARCHAR2	Document or stop *word* from stoplist object to be tokenized. If the document is larger than 32512 bytes then Oracle Text will report a document level indexing error.
2	IN OUT	VARCHAR2	Tokens encoded as XML. If the document contains no tokens, then either NULL must be returned or the tokens element in the XML document returned must contain no child elements. Byte length of the data must be less than or equal to 32512. To improve performance, use the NOCOPY hint when declaring this parameter. This passes the data by reference, rather than passing data by value. The XML document returned by this procedure should not include unnecessary whitespace characters (typically used to improve readability). This reduces the size of the XML document which in turn minimizes the transfer time. To improve performance, index_procedure should not validate the XML document with the corresponding XML schema at run-time. Note that this parameter is IN OUT for performance purposes. The stored procedure has no need to use the IN value.
3	IN	BOOLEAN	Oracle Text sets this parameter to TRUE when Oracle Text needs the character offset and character length of the tokens as found in the document being tokenized. Oracle Text sets this parameter to FALSE when Text is not interested in the character offset and character length of the tokens as found in the document being tokenized. This implies that the XML attributes off and len must not be used.

CLOB Interface Table 2–27 describes the CLOB interface that enables the document or stop word from stoplist object to be tokenized to be passed as CLOB from Oracle Text to the stored procedure and for the tokens to be passed as CLOB as well from the stored procedure back to Oracle Text.

The user-defined lexer indexing procedure should use this interface when at least one of the documents in the column to be indexed is larger than 32512 bytes or the corresponding tokens are represented by more than 32512 bytes.

Table 2–27 CLOB Interface for INDEX_PROCEDURE

Parameter Position	Parameter Mode	Parameter Datatype	Description
1	IN	CLOB	Document or stop *word* from stoplist object to be tokenized.
2	IN OUT	CLOB	Tokens encoded as XML. If the document contains no tokens, then either NULL must be returned or the tokens element in the XML document returned must contain no child elements. To improve performance, use the NOCOPY hint when declaring this parameter. This passes the data by reference, rather than passing data by value. The XML document returned by this procedure should not include unnecessary whitespace characters (typically used to improve readability). This reduces the size of the XML document which in turn minimizes the transfer time. To improve performance, index_procedure should not validate the XML document with the corresponding XML schema at run-time. Note that this parameter is IN OUT for performance purposes. The stored procedure has no need to use the IN value. The IN value will always be a truncated CLOB.
3	IN	BOOLEAN	Oracle Text sets this parameter to TRUE when Oracle Text needs the character offset and character length of the tokens as found in the document being tokenized. Oracle Text sets this parameter to FALSE when Text is not interested in the character offset and character length of the tokens as found in the document being tokenized. This implies that the XML attributes off and len must not be used.

The first and second parameters are temporary CLOBS. Avoid assigning these CLOB locators to other locator variables. Assigning the formal parameter CLOB locator to another locator variable causes a new copy of the temporary CLOB to be created resulting in a performance hit.

QUERY_PROCEDURE

This callback stored procedure is called by Oracle Text as needed to tokenize *word*s in the query. A space-delimited group of characters (excluding the query operators) in the query will be identified by Oracle Text as a *word*.

Requirements This procedure can be a PL/SQL stored procedure.

The index owner must have EXECUTE privilege on this stored procedure.

This stored procedure must not be replaced or be dropped after the index is created. You can replace or drop this stored procedure after the index is dropped.

Restrictions This procedure must not perform any of the following operations:

- Rollback
- Explicitly or implicitly commit the current transaction
- Enter any other transaction control statement

- Alter the session language or territory

The child elements of the root element tokens of the XML document returned must be in the same order as the tokens occur in the query *word* being tokenized.

The behavior of this stored procedure must be deterministic with respect to all parameters.

Parameters Table 2–28 describes the interface for the user-defined lexer query procedure:

Table 2–28 User-defined Lexer Query Procedure XML Schema Attributes

Parameter Position	Parameter Mode	Parameter Datatype	Description
1	IN	VARCHAR2	Query *word* to be tokenized.
2	IN	CTX_ULEXER.WILDCARD_TAB	Character offsets of wildcard characters (% and _) in the query *word*. If the query *word* passed in by Oracle Text does not contain any wildcard characters then this index-by table will be empty. The wildcard characters in the query *word* must be preserved in the tokens returned in order for the wildcard query feature to work properly. The character offset is 0 (zero) based. Offset information follows USC-2 codepoint semantics.
3	IN OUT	VARCHAR2	Tokens encoded as XML. If the query *word* contains no tokens then either NULL must be returned or the tokens element in the XML document returned must contain no child elements. The length of the data must be less-than or equal to 32512 bytes.

Encoding Tokens as XML

The sequence of tokens returned by your stored procedure must be represented as an XML 1.0 document. The XML document must be valid with respect to the XML Schemas given in the following sections.

- XML Schema for No-Location, User-defined Indexing Procedure
- XML Schema for User-defined Indexing Procedure with Location
- XML Schema for User-defined Lexer Query Procedure

Limitations To boost performance of this feature, the XML parser in Oracle Text will not perform validation and will not be a full-featured XML compliant parser. This implies that only minimal XML features will be supported. The following XML features are not supported:

- Document Type Declaration (for example, `<!DOCTYPE [...]>`) and therefore entity declarations. Only the following built-in entities can be referenced: lt, gt, amp, quot, and apos.
- CDATA sections.
- Comments.

- Processing Instructions.
- XML declaration (for example, `<?xml version="1.0" ...?>`).
- Namespaces.
- Use of elements and attributes other than those defined by the corresponding XML Schema.
- Character references (for example ট).
- xml:space attribute.
- xml:lang attribute

XML Schema for No-Location, User-defined Indexing Procedure

This section describes additional constraints imposed on the XML document returned by the user-defined lexer indexing procedure when the third parameter is `FALSE`. The XML document returned must be valid with respect to the following XML Schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="tokens">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="eos" type="EmptyTokenType"/>
          <xsd:element name="eop" type="EmptyTokenType"/>
          <xsd:element name="num" type="xsd:token"/>
          <xsd:group ref="IndexCompositeGroup"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!--
  Enforce constraint that compMem element must be preceeded by word element
  or compMem element for indexing
  -->
  <xsd:group name="IndexCompositeGroup">
    <xsd:sequence>
      <xsd:element name="word" type="xsd:token"/>
      <xsd:element name="compMem" type="xsd:token" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:group>

  <!-- EmptyTokenType defines an empty element without attributes -->
  <xsd:complexType name="EmptyTokenType"/>

</xsd:schema>
```

Here are some of the constraints imposed by this XML Schema:

- The root element is tokens. This is mandatory. It has no attributes.
- The root element can have zero or more child elements. The child elements can be one of the following elements: eos, eop, num, word, and compMem. Each of these represent a specific type of token.
- The compMem element must be preceded by a word element or a compMem element.

- The eos and eop elements have no attributes and must be empty elements.
- The num, word, and compMem elements have no attributes. Oracle Text will normalize the content of these elements as follows: convert whitespace characters to space characters, collapse adjacent space characters to a single space character, remove leading and trailing spaces, perform entity reference replacement, and truncate to 64 bytes.

Table 2–29 describes the element names defined in the preceding XML Schema.

Table 2–29 User-defined Lexer Indexing Procedure XML Schema Element Names

Element	Description
word	This element represents a simple word token. The content of the element is the word itself. Oracle Text does the work of identifying this token as being a stop word or non-stop word and processing it appropriately.
num	This element represents an arithmetic number token. The content of the element is the arithmetic number itself. Oracle Text treats this token as a stop word if the stoplist preference has `NUMBERS` added as the stopclass. Otherwise this token is treated the same way as the word token. Supporting this token type is optional. Without support for this token type, adding the `NUMERBS` stopclass will have no effect.
eos	This element represents end-of-sentence token. Oracle Text uses this information so that it can support `WITHIN SENTENCE` queries. Supporting this token type is optional. Without support for this token type, queries against the `SENTENCE` section will not work as expected.
eop	This element represents end-of-paragraph token. Oracle Text uses this information so that it can support `WITHIN PARAGRAPH` queries. Supporting this token type is optional. Without support for this token type, queries against the `PARAGRAPH` section will not work as expected.
compMem	Same as the word element, except that the implicit word offset is the same as the previous word token. Support for this token type is optional.

Example Document: Vom Nordhauptbahnhof und aus der Innenstadt zum Messegelände.

Tokens:

```
<tokens>
  <word> VOM </word>
  <word> NORDHAUPTBAHNHOF </word>
  <compMem>NORD</compMem>
  <compMem>HAUPT </compMem>
  <compMem>BAHNHOF </compMem>
  <compMem>HAUPTBAHNHOF </compMem>
  <word> UND </word>
  <word> AUS </word>
  <word> DER </word>
  <word> INNENSTADT </word>
  <word> ZUM </word>
  <word> MESSEGELÄNDE </word>
  <eos/>
</tokens>
```

Example Document: Oracle Database 11*g* Release 1

Tokens:

```
<tokens>
  <word> ORACLE11G</word>
  <word> RELEASE </word>
  <num> 1 </num>
</tokens>
```

Example Document: WHERE salary<25000.00 AND job = 'F&B Manager'

Tokens:

```
<tokens>
  <word> WHERE </word>
  <word> salary&lt;2500.00 </word>
  <word> AND </word>
  <word> job </word>
  <word> F&amp;B </word>
  <word> Manager </word>
</tokens>
```

XML Schema for User-defined Indexing Procedure with Location

This section describes additional constraints imposed on the XML document returned by the user-defined lexer indexing procedure when the third parameter is `TRUE`. The XML document returned must be valid according to the following XML schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="tokens">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="eos" type="EmptyTokenType"/>
          <xsd:element name="eop" type="EmptyTokenType"/>
          <xsd:element name="num" type="DocServiceTokenType"/>
          <xsd:group ref="DocServiceCompositeGroup"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!--
  Enforce constraint that compMem element must be preceeded by word element
  or compMem element for document service
  -->
  <xsd:group name="DocServiceCompositeGroup">
    <xsd:sequence>
      <xsd:element name="word" type="DocServiceTokenType"/>
      <xsd:element name="compMem" type="DocServiceTokenType" minOccurs="0"
           maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:group>

  <!-- EmptyTokenType defines an empty element without attributes -->
  <xsd:complexType name="EmptyTokenType"/>

  <!--
  DocServiceTokenType defines an element with content and mandatory attributes
  -->
  <xsd:complexType name="DocServiceTokenType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:token">
```

```
        <xsd:attribute name="off" type="OffsetType" use="required"/>
        <xsd:attribute name="len" type="xsd:unsignedShort" use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

  <xsd:simpleType name="OffsetType">
    <xsd:restriction base="xsd:unsignedInt">
      <xsd:maxInclusive value="2147483647"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

Some of the constraints imposed by this XML Schema are as follows:

- The root element is tokens. This is mandatory. It has no attributes.
- The root element can have zero or more child elements. The child elements can be one of the following elements: eos, eop, num, word, and compMem. Each of these represent a specific type of token.
- The compMem element must be preceded by a word element or a compMem element.
- The eos and eop elements have no attributes and must be empty elements.
- The num, word, and compMem elements have two mandatory attributes: `off` and len. Oracle Text will normalize the content of these elements as follows: convert whitespace characters to space characters, collapse adjacent space characters to a single space character, remove leading and trailing spaces, perform entity reference replacement, and truncate to 64 bytes.
- The `off` attribute value must be an integer between 0 and 2147483647 inclusive.
- The `len` attribute value must be an integer between 0 and 65535 inclusive.

Table 2–29 describes the element types defined in the preceding XML Schema.

Table 2–30 describes the attributes defined in the preceding XML Schema.

Table 2–30 User-defined Lexer Indexing Procedure XML Schema Attributes

Attribute	Description
off	This attribute represents the character offset of the token as it appears in the document being tokenized. The offset is with respect to the character document passed to the user-defined lexer indexing procedure, not the document fetched by the datastore. The document fetched by the datastore may be pre-processed by the filter object or the section group object, or both, before being passed to the user-defined lexer indexing procedure. The offset of the first character in the document being tokenized is 0 (zero). Offset information follows USC-2 codepoint semantics.

Table 2–30 (Cont.) User-defined Lexer Indexing Procedure XML Schema Attributes

Attribute	Description
len	This attribute represents the character length (same semantics as SQL function `LENGTH`) of the token as it appears in the document being tokenized. The length is with respect to the character document passed to the user-defined lexer indexing procedure, not the document fetched by the datastore. The document fetched by the datastore may be pre-processed by the filter object or the section group object before being passed to the user-defined lexer indexing procedure. Length information follows USC-2 codepoint semantics.

Sum of `off` attribute value and `len` attribute value must be less than or equal to the total number of characters in the document being tokenized. This is to ensure that the document offset and characters being referenced are within the document boundary.

Example Document: User-defined Lexer.

Tokens:

```
<tokens>
  <word off="0" len="4"> USE </word>
  <word off="5" len="7"> DEF </word>
  <word off="13" len="5"> LEX </word>
  <eos/>
</tokens>
```

XML Schema for User-defined Lexer Query Procedure

This section describes additional constraints imposed on the XML document returned by the user-defined lexer query procedure. The XML document returned must be valid with respect to the following XML Schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="tokens">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="num" type="QueryTokenType"/>
          <xsd:group ref="QueryCompositeGroup"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

<!--
Enforce constraint that compMem element must be preceeded by word element
or compMem element for query
-->
  <xsd:group name="QueryCompositeGroup">
    <xsd:sequence>
      <xsd:element name="word" type="QueryTokenType"/>
      <xsd:element name="compMem" type="QueryTokenType" minOccurs="0"
                                              maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:group>

  <!--
```

```
  QueryTokenType defines an element with content and with an optional attribute
  -->
  <xsd:complexType name="QueryTokenType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:token">
        <xsd:attribute name="wildcard" type="WildcardType" use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

  <xsd:simpleType name="WildcardType">
    <xsd:restriction base="WildcardBaseType">
      <xsd:minLength value="1"/>
      <xsd:maxLength value="64"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="WildcardBaseType">
    <xsd:list>
      <xsd:simpleType>
        <xsd:restriction base="xsd:unsignedShort">
          <xsd:maxInclusive value="378"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:list>
  </xsd:simpleType>

</xsd:schema>
```

Here are some of the constraints imposed by this XML Schema:

- The `root` element is tokens. This is mandatory. It has no attributes.
- The `root` element can have zero or more child elements. The child elements can be one of the following elements: `num` and `word`. Each of these represent a specific type of token.
- The `compMem` element must be preceded by a `word` element or a `compMem` element.

  The purpose of `compMem` is to enable `USER_LEXER` queries to return multiple forms for a single query. For example, if a user-defined lexer indexes the word bank as `BANK(FINANCIAL)` and `BANK(RIVER)`, the query procedure can return the first term as a `word` and the second as a `compMem` element:

  ```
  <tokens>
    <word>BANK(RIVER)</word>
    <compMem>BANK(FINANCIAL)</compMem>
  </tokens>
  ```

  See Table 2–31, " User-defined Lexer Query Procedure XML Schema Attributes" on page 2-56 for more on the `compMem` element.
- The num and word elements have a single optional attribute: wildcard. Oracle Text will normalize the content of these elements as follows: convert whitespace characters to space characters, collapse adjacent space characters to a single space character, remove leading and trailing spaces, perform entity reference replacement, and truncate to 64 bytes.
- The wildcard attribute value is a white-space separated list of integers. The minimum number of integers is 1 and the maximum number of integers is 64. The

value of the integers must be between 0 and 378 inclusive. The intriguers in the list can be in any order.

Table 2–29 describes the element types defined in the preceding XML Schema.

Table 2–31 describes the attribute defined in the preceding XML Schema.

Table 2–31 User-defined Lexer Query Procedure XML Schema Attributes

Attribute	Description
compMem	Same as the word element, but its implicit word offset is the same as the previous word token. Oracle Text will equate this token with the previous word token and with subsequent compMem tokens using the query EQUIV operator.
wildcard	Any% or _ characters in the query which are not escaped by the user are considered wildcard characters because they are replaced by other characters. These wildcard characters in the query must be preserved during tokenization in order for the wildcard query feature to work properly. This attribute represents the character offsets (same semantics as SQL function LENGTH) of wildcard characters in the content of the element. Oracle Text will adjust these offsets for any normalization performed on the content of the element. The characters pointed to by the offsets must either be% or _ characters. The offset of the first character in the content of the element is 0. Offset information follows USC-2 codepoint semantics. If the token does not contain any wildcard characters then this attribute must not be specified.

Example Query *word*: pseudo-%morph%

Tokens:

```
<tokens>
  <word> PSEUDO </word>
  <word wildcard="1 7"> %MORPH% </word>
</tokens>
```

Example Query *word*: <%>

```
Tokens:
<tokens>
  <word wildcard="5"> &lt;%&gt; </word>
</tokens>
```

WORLD_LEXER

Use the WORLD_LEXER to index text columns that contain documents of different languages. For example, use this lexer to index a text column that stores English, Japanese, and German documents.

WORLD_LEXER differs from MULTI_LEXER in that WORLD_LEXER automatically detects the language(s) of a document. Unlike MULTI_LEXER, WORLD_LEXER does not require you to have a language column in your base table nor to specify the language column when you create the index. Moreover, it is not necessary to use sub-lexers, as with MULTI_LEXER. (See "MULTI_LEXER" on page 2-38.)

WORLD_LEXER supports all database character sets, and for languages whose character sets are Unicode-based, it supports the Unicode 5.0 standard. For a list of languages that WORLD_LEXER can work with, see "World Lexer Features" on page D-4.

WORLD_LEXER Attribute

The WORLD_VGRAM_LEXER has the following attribute:

Table 2–32 WORLD_LEXER Attribute

Attribute	Attribute Value
mixed_case	Enable mixed-case (upper- and lower-case) searches of text (for example, *cat* and *Cat*). Allowable values are YES and NO (default).

WORLD_LEXER Example

Here is an example of creating an index using WORLD_LEXER.

```
exec ctx_ddl.create_preference('MYLEXER', 'world_lexer');
create index doc_idx on doc(data)
  indextype is CONTEXT
  parameters ('lexer MYLEXER
               stoplist CTXSYS.EMPTY_STOPLIST');
```

Wordlist Type

Use the wordlist preference to enable the query options such as stemming, fuzzy matching for your language. You can also use the wordlist preference to enable substring and prefix indexing, which improves performance for wildcard queries with CONTAINS and CATSEARCH.

To create a wordlist preference, you must use BASIC_WORDLIST, which is the only type available.

BASIC_WORDLIST

Use BASIC_WORDLIST type to enable stemming and fuzzy matching or to create prefix indexes with Text indexes.

See Also: Chapter 3, "Oracle Text CONTAINS Query Operators"

Table 2–33 lists the attributes for BASIC_WORDLIST.

Table 2–33 BASIC_WORDLIST Attributes

Attribute	Attribute Values
stemmer	Specify which language stemmer to use. You can specify one of the following stemmers: NULL (no stemming) ENGLISH (English inflectional) DERIVATIONAL (English derivational) DUTCH FRENCH GERMAN ITALIAN SPANISH AUTO (Automatic language-detection for stemming, derived from the database session language. If the database session language is AMERICAN or ENGLISH, then the ENGLISH stemmer is used. Does not auto-detect JAPANESE.) JAPANESE
fuzzy_match	Specify which fuzzy matching cluster to use. You can specify one of the following types: AUTO (Automatic language detection for stemming.) CHINESE_VGRAM DUTCH ENGLISH FRENCH GENERIC GERMAN ITALIAN JAPANESE_VGRAM KOREAN OCR SPANISH
fuzzy_score	Specify a default lower limit of fuzzy score. Specify a number between 0 and 80. Text with scores below this number is not returned. Default is 60.
fuzzy_numresults	Specify the maximum number of fuzzy expansions. Use a number between 0 and 5,000. Default is 100.
substring_index	Specify TRUE for Oracle Text to create a substring index. A substring index improves left-truncated and double-truncated wildcard queries such as *%ing* or *%benz%*. Default is FALSE.
prefix_index	Specify TRUE to enable prefix indexing. Prefix indexing improves performance for right truncated wildcard searches such as TO%. Default is FALSE.
prefix_min_length	Specify the minimum length of indexed prefixes. Default is 1. Length information must follow USC-2 codepoint semantics.
prefix_max_length	Specify the maximum length of indexed prefixes. Default is 64. Length information must follow USC-2 codepoint semantics.

Table 2–33 (Cont.) BASIC_WORDLIST Attributes

Attribute	Attribute Values
wildcard_maxterms	Specify the maximum number of terms in a wildcard expansion. The maximum value is 50000 and the default value is 20000. If you specify a value of 0, then the number of wildcard expansions will be unbounded. Note that when set to 0, the system may run out of memory due to the high number of wildcard expansions.
ndata_base_letter	Specify whether characters that have diacritical marks are converted to their base form before being stored in the Text index or queried by the NDATA operator. FALSE (default) or TRUE When set to FALSE, no base lettering is used.
ndata_alternate_spelling	Specify whether to enable alternate spelling for German, Danish, and Swedish. Enabling alternate spelling allows you to index NDATA section data and query using the NDATA operator in alternate form. FALSE (default) or TRUE When set to FALSE, no alternate spelling is used.
ndata_thesaurus	Name of the thesaurus used for alternate name expansion.
ndata_join_particles	A list of colon-separated name particles that can be joined with a name that follows them.

stemmer

Specify the stemmer used for word stemming in Text queries. When you do not specify a value for STEMMER, the default is ENGLISH.

Specify AUTO for the system to automatically set the stemming language according to the language setting of the database session. If the database language is AMERICAN or ENGLISH, then the ENGLISH stemmer is automatically used. Otherwise, the stemmer that maps to the database session language is used.

When there is no stemmer for a language, the default is NULL. With the NULL stemmer, the stem operator is ignored in queries.

You can create your own stemming user-dictionary. See "Stemming User-Dictionaries" on page 2-35 for more information.

Note: The STEMMER attribute of BASIC_WORDLIST preference will be ignored if the database session language causes MULTI_LEXER to choose a SUB_LEXER with the same setting as wildcard_maxterms or ndata_base_letter.

In this case, the same stemmer that is used by the BASIC_LEXER during indexing will be used to determine the stem of the query term during query.

---

fuzzy_match

Specify which fuzzy matching routines are used for the column. Fuzzy matching is currently supported for English, Japanese, and, to a lesser extent, the Western European languages.

> **Note:** The `fuzzy_match` attributes value for Chinese and Korean are dummy attribute values that prevent the English and Japanese fuzzy matching routines from being used on Chinese and Korean text.

The default for `fuzzy_match` is `GENERIC`.

Specify `AUTO` for the system to automatically set the fuzzy matching language according to language setting of the session.

fuzzy_score
Specify a default lower limit of fuzzy score. Specify a number between 0 and 80. Text with scores below this number are not returned. The default is 60.

Fuzzy score is a measure of how close the expanded word is to the query word. The higher the score the better the match. Use this parameter to limit fuzzy expansions to the best matches.

fuzzy_numresults
Specify the maximum number of fuzzy expansions. Use a number between 0 and 5000. The default is 100.

Setting a fuzzy expansion limits the expansion to a specified number of the best matching words.

substring_index
Specify `TRUE` for Oracle Text to create a substring index. A substring index improves performance for left-truncated or double-truncated wildcard queries such as *%ing* or *%benz%*. The default is false.

Substring indexing has the following impact on indexing and disk resources:

- Index creation and DML processing is up to 4 times slower
- Index creation with `substring_index` enabled requires more rollback segments during index flushes than with substring index off. Oracle recommends that you do either of the following when creating a substring index:
  - Make available double the usual rollback or
  - Decrease the index memory to reduce the size of the index flushes to disk

prefix_index
Specify `yes` to enable prefix indexing. Prefix indexing improves performance for right truncated wildcard searches such as *TO%*. Default is `NO`.

> **Note:** Enabling prefix indexing increases index size.

Prefix indexing chops up tokens into multiple prefixes to store in the $I table. For example, words `TOKEN` and `TOY` are normally indexed as follows in the $I table:

Token	Type	Information
TOKEN	0	DOCID 1 POS 1
TOY	0	DOCID 1 POS 3

With prefix indexing, Oracle Text indexes the prefix substrings of these tokens as follows with a new token type of 6:

Token	Type	Information
TOKEN	0	DOCID 1 POS 1
TOY	0	DOCID 1 POS 3
T	6	DOCID 1 POS 1 POS 3
TO	6	DOCID 1 POS 1 POS 3
TOK	6	DOCID 1 POS 1
TOKE	6	DOCID 1 POS 1
TOKEN	6	DOCID 1 POS 1
TOY	6	DOCID 1 POS 3

Wildcard searches such as `TO%` are now faster because Oracle Text does no expansion of terms and merging of result sets. To obtain the result, Oracle Text need only examine the (TO,6) row.

prefix_min_length
Specify the minimum length of indexed prefixes. Default is 1.

For example, setting `prefix_min_length` to 3 and `prefix_max_length` to 5 indexes all prefixes between 3 and 5 characters long.

> **Note:** A wildcard search whose pattern is below the minimum length or above the maximum length is searched using the slower method of equivalence expansion and merging.

prefix_max_length
Specify the maximum length of indexed prefixes. Default is 64.

For example, setting `prefix_min_length` to 3 and `prefix_max_length` to 5 indexes all prefixes between 3 and 5 characters long.

> **Note:** A wildcard search whose pattern is below the minimum length or above the maximum length is searched using the slower method of equivalence expansion and merging.

wildcard_maxterms
Specify the maximum number of terms in a wildcard (%) expansion. Use this parameter to keep wildcard query performance within an acceptable limit. Oracle Text returns an error when the wildcard query expansion exceeds this number.

ndata_base_letter
Specify whether characters that have diacritical marks (umlauts, cedillas, acute accents, and so on) are converted to their base form before being stored in the Text index or queried by the `NDATA` operator. The default is `FALSE` (base-letter conversion disabled). For more information on base-letter conversions, see "Base-Letter Conversion" on page 15-2.

ndata_alternate_spelling
Specify whether to enable alternate spelling for German, Danish, and Swedish. Enabling alternate spelling allows you to index NDATA section data and query using the NDATA operator in alternate form.

When ndata_base_letter is enabled at the same time as ndata_alternate_spelling, NDATA section data is serially transformed first by alternate spelling and then by base lettering. For more information about the alternate spelling conventions Oracle Text uses, see "Alternate Spelling" on page 15-2.

ndata_thesaurus
Specify a name of the thesaurus used for alternate name expansion. The indexing engine expands names in documents using synonym rings in the thesaurus. A user should make use of homographic disambiguating feature of the thesaurus to distinguish common nicknames.

An example is:

```
Albert
  SYN Al
  SYN Bert
Alfred
  SYN Al
  SYN Fred
```

A simple definition such as the above will put Albert, Alfred, Al, Bert, and Fred into the same synonym ring. This will cause an unexpected expansion such that the expansion of Bert includes Fred. To prevent this, you can use homographic disambiguation as in:

```
Albert
  SYN Al (Albert)
  SYN Bert (Albert)
Alfred
  SYN Al (Alfred)
  SYN Fred (Alfred)
```

This forms two synonym rings, Albert-Al-Bert and Alfred-Al-Fred. Thus, the expansion of Bert no longer includes Fred. A more detailed example is:

```
begin
  ctx_ddl.create_preference('NDAT_PREF', 'BASIC_WORDLIST');
  ctx_ddl.set_attribute('NDATA_PREF', 'NDATA_ALTERNATE_SPELLING', 'FALSE');
  ctx_ddl.set_attribute('NDATA_PREF', 'NDATA_BASE_LETTER', 'TRUE');
  ctx_ddl.set_attribute('NDATA_PREF', 'NDATA_THESAURUS', 'NICKNAMES');
end;
```

Note: A sample thesaurus for names can be found in the $ORACLE_HOME/ctx/sample/thes directory. This file is dr0thsnames.txt.

ndata_join_particles
Specify a list of colon-separated name particles that can be joined with a name that follows them. A name particle, such as da, is written separately from or joined with its following name like da Vinci or daVinci. The indexing engine generates index data for both separated and join versions of a name when it finds a name particle specified in this prefence. The same happens in the query processing for better recall.

BASIC_WORDLIST Example

The following example shows the use of the BASIC_WORDLIST type.

Enabling Fuzzy Matching and Stemming

The following example enables stemming and fuzzy matching for English. The preference STEM_FUZZY_PREF sets the number of expansions to the maximum allowed. This preference also instructs the system to create a substring index to improve the performance of double-truncated searches.

```
begin
  ctx_ddl.create_preference('STEM_FUZZY_PREF', 'BASIC_WORDLIST');
  ctx_ddl.set_attribute('STEM_FUZZY_PREF','FUZZY_MATCH','ENGLISH');
  ctx_ddl.set_attribute('STEM_FUZZY_PREF','FUZZY_SCORE','0');
  ctx_ddl.set_attribute('STEM_FUZZY_PREF','FUZZY_NUMRESULTS','5000');
  ctx_ddl.set_attribute('STEM_FUZZY_PREF','SUBSTRING_INDEX','TRUE');
  ctx_ddl.set_attribute('STEM_FUZZY_PREF','STEMMER','ENGLISH');
end;
```

To create the index in SQL, enter the following statement:

```
create index fuzzy_stem_subst_idx on mytable ( docs )
  indextype is ctxsys.context parameters ('Wordlist STEM_FUZZY_PREF');
```

Enabling Sub-string and Prefix Indexing

The following example sets the wordlist preference for prefix and sub-string indexing. For prefix indexing, it specifies that Oracle Text create token prefixes between 3 and 4 characters long:

```
begin
ctx_ddl.create_preference('mywordlist', 'BASIC_WORDLIST');
ctx_ddl.set_attribute('mywordlist','PREFIX_INDEX','TRUE');
ctx_ddl.set_attribute('mywordlist','PREFIX_MIN_LENGTH',3);
ctx_ddl.set_attribute('mywordlist','PREFIX_MAX_LENGTH', 4);
ctx_ddl.set_attribute('mywordlist','SUBSTRING_INDEX', 'YES');
end
```

Setting Wildcard Expansion Limit

Use the wildcard_maxterms attribute to set the maximum allowed terms in a wildcard expansion.

```
--- create a sample table
drop table quick ;
create table quick
  (
    quick_id number primary key,
    text      varchar(80)
  );

--- insert a row with 10 expansions for 'tire%'
insert into quick ( quick_id, text )
  values ( 1, 'tire tirea tireb tirec tired tiree tiref tireg tireh tirei tirej');
commit;

--- create an index using wildcard_maxterms=100
begin
    Ctx_Ddl.Create_Preference('wildcard_pref', 'BASIC_WORDLIST');
    ctx_ddl.set_attribute('wildcard_pref', 'wildcard_maxterms', 100) ;
end;
```

```
/
create index wildcard_idx on quick(text)
    indextype is ctxsys.context
    parameters ('Wordlist wildcard_pref') ;

--- query on 'tire%' - should work fine
select quick_id from quick
  where contains ( text, 'tire%' ) > 0;

--- now re-create the index with wildcard_maxterms=5

drop index wildcard_idx ;

begin
    Ctx_Ddl.Drop_Preference('wildcard_pref');
    Ctx_Ddl.Create_Preference('wildcard_pref', 'BASIC_WORDLIST');
    ctx_ddl.set_attribute('wildcard_pref', 'wildcard_maxterms', 5) ;
end;
/

create index wildcard_idx on quick(text)
    indextype is ctxsys.context
    parameters ('Wordlist wildcard_pref') ;

--- query on 'tire%' gives "wildcard query expansion resulted in too many terms"
select quick_id from quick
  where contains ( text, 'tire%' ) > 0;
```

Storage Types

Use the storage preference to specify tablespace and creation parameters for tables associated with a Text index. The system provides a single storage type called BASIC_STORAGE:

Table 2–34 Storage Types

Type	Description
BASIC_STORAGE	Indexing type used to specify the tablespace and creation parameters for the database tables and indexes that constitute a Text index.

BASIC_STORAGE

The BASIC_STORAGE type specifies the tablespace and creation parameters for the database tables and indexes that constitute a Text index.

The clause you specify is added to the internal CREATE TABLE (CREATE INDEX for the i_index_clause) statement at index creation. You can specify most allowable clauses, such as storage, LOB storage, or partitioning. However, you cannot specify an index organized table clause.

> **See Also:** For more information about how to specify CREATE TABLE and CREATE INDEX statements, see *Oracle Database SQL Language Reference*.

BASIC_STORAGE has the following attributes:

Table 2–35 BASIC_STORAGE Attributes

Attribute	Attribute Value
i_index_clause	Parameter clause for dr$*indexname*$X index creation. Specify storage and tablespace clauses to add to the end of the internal CREATE INDEX statement. The default clause is: 'COMPRESS 2' which instructs Oracle Text to compress this index table. If you choose to override the default, Oracle recommends including COMPRESS 2 in your parameter clause to compress this table, because such compression saves disk space and helps query performance.
i_rowid_index_clause	Parameter clause to specify the storage clause for the $R index on dr$*rowid* column of the $I table. Specify storage and tablespace clauses to add to the end of the internal CREATE INDEX statement. This clause is only used by the CTXCAT index type.
i_table_clause	Parameter clause for dr$*indexname*$I table creation. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement. The I table is the index data table. **Note:** Oracle strongly recommends that you do not specify "disable storage in row" for $I LOBs, as this will greatly degrade the query performance.
k_table_clause	Parameter clause for dr$*indexname*$K table creation. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement. The K table is the keymap table.
r_table_clause	Parameter clause for dr$*indexname*$R table creation. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement. The R table is the rowid table. The default clause is: 'LOB(DATA) STORE AS (CACHE)'. If you modify this attribute, always include this clause for good performance.
n_table_clause	Parameter clause for dr$*indexname*$N table creation. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement. The N table is the negative list table.
p_table_clause	Parameter clause for the substring index if you have enabled SUBSTRING_INDEX in the BASIC_WORDLIST. Specify storage and tablespace clauses to add to the end of the internal CREATE INDEX statement. The P table is an index-organized table so the storage clause you specify must be appropriate to this type of table.

Table 2–35 (Cont.) BASIC_STORAGE Attributes

Attribute	Attribute Value
s_table_clause	Parameter clause for dr$*indexname*$S table creation*. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement. The default clause is nocompress. * For performance reasons, $S table must be created on a tablespace with db block size >= 4K without overflow segment and without a PCTTHRESHOLD clause. If $S is created on a tablespace with db block size < 4K, or is created with an overflow segment or with PCTTHRESHOLD clause, then appropriate errors will be raised during CREATE INDEX. The S table is the table that stores SDATA section values. If this clause is specified for a storage preference in an index without SDATA, then it will have no effect on the index, and index creation will still succeed.

Storage Default Behavior

By default, BASIC_STORAGE attributes are not set. In such cases, the Text index tables are created in the index owner's default tablespace. Consider the following statement, entered by user IUSER, with no BASIC_STORAGE attributes set:

```
create index IOWNER.idx on TOWNER.tab(b) indextype is ctxsys.context;
```

In this example, the text index is created in IOWNER's default tablespace.

Storage Examples

The following examples specify that the index tables are to be created in the foo tablespace with an initial extent of 1K:

```
begin
ctx_ddl.create_preference('mystore', 'BASIC_STORAGE');
ctx_ddl.set_attribute('mystore', 'I_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'K_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'R_TABLE_CLAUSE',
                      'tablespace users storage (initial 1K) lob
                       (data) store as (disable storage in row cache)');
ctx_ddl.set_attribute('mystore', 'N_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'I_INDEX_CLAUSE',
                      'tablespace foo storage (initial 1K) compress 2');
ctx_ddl.set_attribute('mystore', 'P_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'S_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
end;
```

Section Group Types

To enter WITHIN queries on document sections, you must create a section group before you define your sections. Specify your section group in the parameter clause of CREATE INDEX.

To create a section group, you can specify one of the following group types with the CTX_DDL.CREATE_SECTION_GROUP procedure:

Table 2–36 Section Group Types

Type	Description
NULL_SECTION_GROUP	Use this group type when you define no sections or when you define *only* SENTENCE or PARAGRAPH sections. This is the default.
BASIC_SECTION_GROUP	Use this group type for defining sections where the start and end tags are of the form <A> and </A>. Note: This group type does not support input such as unbalanced parentheses, comments tags, and attributes. Use HTML_SECTION_GROUP for this type of input.
HTML_SECTION_GROUP	Use this group type for indexing HTML documents and for defining sections in HTML documents.
XML_SECTION_GROUP	Use this group type for indexing XML documents and for defining sections in XML documents. All sections to be indexed must be manually defined for this group.
AUTO_SECTION_GROUP	Use this group type to automatically create a zone section for each start-tag/end-tag pair in an XML document. The section names derived from XML tags are case sensitive as in XML. Attribute sections are created automatically for XML tags that have attributes. Attribute sections are named in the form tag@attribute. Special sections can be added to AUTO_SECTION_GROUP for WITHIN SENTENCE and WITHIN PARAGRAPH searches. Once a sentence or paragraph section is added to the AUTO_SECTION_GROUP, sections with corresponding tag names 'sentence' or 'paragraph' (case insensitive) are treated as stop sections. Stop sections, empty tags, processing instructions, and comments are not indexed. The following limitations apply to automatic section groups: ■ You cannot add zone, field, sdata, or special sections to an automatic section group. ■ You can define a stop section that applies only to one particular type; that is, if you have two different XML DTDs, both of which use a tag called FOO, you can define (TYPE1)FOO to be stopped, but (TYPE2)FOO to not be stopped. ■ The length of the indexed tags, including prefix and namespace, cannot exceed 64 bytes. Tags longer than this are not indexed.
PATH_SECTION_GROUP	Use this group type to index XML documents. Behaves like the AUTO_SECTION_GROUP. The difference is that with this section group you can do path searching with the INPATH and HASPATH operators. Queries are also case-sensitive for tag and attribute names. Stop sections are not allowed.
NEWS_SECTION_GROUP	Use this group for defining sections in newsgroup formatted documents according to RFC 1036.

Section Group Examples

This example shows the use of section groups in both HTML and XML documents.

Creating Section Groups in HTML Documents

The following statement creates a section group called `htmgroup` with the HTML group type.

```
begin
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
end;
```

You can optionally add sections to this group using the procedures in the `CTX_DDL` package, such as `CTX_DDL.ADD_SPECIAL_SECTION` or `CTX_DDL.ADD_ZONE_SECTION`. To index your documents, enter a statement such as:

```
create index myindex on docs(htmlfile) indextype is ctxsys.context
parameters('filter ctxsys.null_filter section group htmgroup');
```

> **See Also:** For more information on section groups, see Chapter 7, "CTX_DDL Package"

Creating Sections Groups in XML Documents

The following statement creates a section group called `xmlgroup` with the `XML_SECTION_GROUP` group type.

```
begin
ctx_ddl.create_section_group('xmlgroup', 'XML_SECTION_GROUP');
end;
```

You can optionally add sections to this group using the procedures in the `CTX_DDL` package, such as `CTX_DDL.ADD_ATTR_SECTION` or `CTX_DDL.ADD_STOP_SECTION`. To index your documents, enter a statement such as:

```
create index myindex on docs(htmlfile) indextype is ctxsys.context
parameters('filter ctxsys.null_filter section group xmlgroup');
```

> **See Also:** For more information on section groups, see Chapter 7, "CTX_DDL Package"

Automatic Sectioning in XML Documents

The following statement creates a section group called `auto` with the `AUTO_SECTION_GROUP` group type. This section group automatically creates sections from tags in XML documents.

```
begin
ctx_ddl.create_section_group('auto', 'AUTO_SECTION_GROUP');
end;

CREATE INDEX myindex on docs(htmlfile) INDEXTYPE IS ctxsys.context
PARAMETERS('filter ctxsys.null_filter section group auto');
```

Classifier Types

This section describes the classifier types used to create a preference for `CTX_CLS.TRAIN` and `CTXRULE` index creation. The following two classifier types are supported:

- RULE_CLASSIFIER
- SVM_CLASSIFIER

Note: In Oracle Database XE Edition, RULE_CLASSIFIER and SVM_CLASSIFIER are not supported because the Data Mining option is not available. This is also true for KMEAN_CLUSTERING.

RULE_CLASSIFIER

Use the RULE_CLASSIFIER type for creating preferences for the query rule generating procedure, CTX_CLS.TRAIN and for CTXRULE creation. The rules generated with this type are essentially query strings and can be easily examined. The queries generated by this classifier can use the AND, NOT, or ABOUT operators. The WITHIN operator is supported for queries on field sections only.

This type has the following attributes:

Table 2–37 RULE_CLASSIFIER Attributes

Attribute	Data Type	Default	Min Value	Max Value	Description
THRESHOLD	I	50	1	99	Specify threshold (in percentage) for rule generation. One rule is output only when its confidence level is larger than threshold.
MAX_TERMS	I	100	20	2000	For each class, a list of relevant terms is selected to form rules. Specify the maximum number of terms that can be selected for each class.
MEMORY_SIZE	I	500	10	4000	Specify memory usage for training in MB. Larger values improve performance.
NT_THRESHOLD	F	0.001	0	0.90	Specify a threshold for term selection. There are two thresholds guiding two steps in selecting relevant terms. This threshold controls the behavior of the first step. At this step, terms are selected as candidate terms for the further consideration in the second step. The term is chosen when the ratio of the occurrence frequency over the number of documents in the training set is larger than this threshold.
TERM_THRESHOLD	I	10	0	100	Specify a threshold as a percentage for term selection. This threshold controls the second step term selection. Each candidate term has a numerical quantity calculated to imply its correlation with a given class. The candidate term will be selected for this class only when the ratio of its quantity value over the maximum value for all candidate terms in the class is larger than this threshold.

Table 2–37 (Cont.) RULE_CLASSIFIER Attributes

Attribute	Data Type	Default	Min Value	Max Value	Description
PRUNE_LEVEL	I	75	0	100	Specify how much to prune a built decision tree for better coverage. Higher values mean more aggressive pruning and the generated rules will have larger coverage but less accuracy.

SVM_CLASSIFIER

Use the SVM_CLASSIFIER type for creating preferences for the rule generating procedure, CTX_CLS.TRAIN, and for CTXRULE creation. This classifier type represents the Support Vector Machine method of classification and generates rules in binary format. Use this classifier type when you need high classification accuracy.

This type has the following attributes:

Table 2–38 SVM_CLASSIFIER Attributes

Attribute Name	Data Type	Default	Min Value	Max Value	Description
MAX_DOCTERMS	I	50	10	8192	Specify the maximum number of terms representing one document.
MAX_FEATURES	I	3,000	1	100,000	Specify the maximum number of distinct features.
THEME_ON	B	FALSE	NULL	NULL	Specify TRUE to use themes as features. Classification with themes requires an installed knowledge base. A knowledge base may or may not have been installed with Oracle Text. For more information on knowledge bases, see the *Oracle Text Application Developer's Guide*.
TOKEN_ON	B	TRUE	NULL	NULL	Specify TRUE to use regular tokens as features.
STEM_ON	B	FALSE	NULL	NULL	Specify TRUE to use stemmed tokens as features. This only works when turning INDEX_STEM on for the lexer.
MEMORY_SIZE	I	500	10	4000	Specify approximate memory size in MB.

Table 2–38 (Cont.) SVM_CLASSIFIER Attributes

Attribute Name	Data Type	Default	Min Value	Max Value	Description
SECTION_WEIGHT	1	2	0	100	Specify the occurrence multiplier for adding a term in a field section as a normal term. For example, by default, the term *cat* in "<A>cat</A>" is a field section term and is treated as a normal term with occurrence equal to 2, but you can specify that it be treated as a normal term with a weight up to 100. SECTION_WEIGHT is only meaningful when the index policy specifies a field section.

Cluster Types

This section describes the cluster types used for creating preferences for the CTX_CLS.CLUSTERING procedure.

Note: In Oracle Database XE Edition, KMEAN_CLUSTERING is not supported because the Data Mining option is not available. This is also true for RULE_CLASSIFIER and SVM_CLASSIFIER.

See Also: For more information about clustering, see "CLUSTERING" in Chapter 6, "CTX_CLS Package" as well as the *Oracle Text Application Developer's Guide*

KMEAN_CLUSTERING

This clustering type has the following attributes:

Table 2–39 KMEAN_CLUSTERING Attributes

Attribute Name	Data Type	Default	Min Value	Max Value	Description
MAX_DOCTERMS	I	50	10	8192	Specify the maximum number of distinct terms representing one document.
MAX_FEATURES	I	3,000	1	500,000	Specify the maximum number of distinct features.
THEME_ON	B	FALSE	NULL	NULL	Specify TRUE to use themes as features. Clustering with themes requires an installed knowledge base. A knowledge base may or may not have been installed with Oracle Text. For more information on knowledge bases, see *Oracle Text Application Developer's Guide*.

Table 2–39 (Cont.) KMEAN_CLUSTERING Attributes

Attribute Name	Data Type	Default	Min Value	Max Value	Description
TOKEN_ON	B	TRUE	NULL	NULL	Specify TRUE to use regular tokens as features.
STEM_ON	B	FALSE	NULL	NULL	Specify TRUE to use stemmed tokens as features. This only works when turning INDEX_STEM on for the lexer.
MEMORY_SIZE	I	500	10	4000	Specify approximate memory size in MB.
SECTION_WEIGHT	1	2	0	100	Specify the occurrence multiplier for adding a term in a field section as a normal term. For example, by default, the term *cat* in "<A>cat</A>" is a field section term and is treated as a normal term with occurrence equal to 2, but you can specify that it be treated as a normal term with a weight up to 100. SECTION_WEIGHT is only meaningful when the index policy specifies a field section.
CLUSTER_NUM	I	200	2	20000	Specify the total number of leaf clusters to be generated.

Stoplists

Stoplists identify the words in your language that are not to be indexed. In English, you can also identify stopthemes that are not to be indexed. By default, the system indexes text using the system-supplied stoplist that corresponds to your database language.

Oracle Text provides default stoplists for most common languages including English, French, German, Spanish, Chinese, Dutch, and Danish. These default stoplists contain only stopwords.

> **See Also:** For more information about the supplied default stoplists, see Appendix E, "Oracle Text Supplied Stoplists"

Multi-Language Stoplists

You can create multi-language stoplists to hold language-specific stopwords. A multi-language stoplist is useful when you use the MULTI_LEXER to index a table that contains documents in different languages, such as English, German.

To create a multi-language stoplist, use the CTX_DLL.CREATE_STOPLIST procedure and specify a stoplist type of MULTI_STOPLIST. Add language specific stopwords with CTX_DDL.ADD_STOPWORD.

At indexing time, the language column of each document is examined, and only the stopwords for that language are eliminated. At query time, the session language setting determines the active stopwords, like it determines the active lexer when using the multi-lexer.

Creating Stoplists

Create your own stoplists using the CTX_DLL.CREATE_STOPLIST procedure. With this procedure you can create a `BASIC_STOPLIST` for single language stoplist, or you can create a `MULTI_STOPLIST` for a multi-language stoplist.

When you create your own stoplist, you must specify it in the parameter clause of `CREATE INDEX`.

To create stoplists for Chinese or Japanese languages, use the `CHINESE_LEXER` or `JAPANESE_LEXER` respectively, and update the appropriate lexicon to be *@contained_such_stopwords*.

Modifying the Default Stoplist

The default stoplist is always named CTXSYS.DEFAULT_STOPLIST. Use the following procedures to modify this stoplist:

- CTX_DDL.ADD_STOPWORD
- CTX_DDL.REMOVE_STOPWORD
- CTX_DDL.ADD_STOPTHEME
- CTX_DDL.ADD_STOPCLASS

When you modify `CTXSYS.DEFAULT_STOPLIST` with the `CTX_DDL` package, you must re-create your index for the changes to take effect.

Dynamic Addition of Stopwords

You can *add* stopwords dynamically to a default or custom stoplist with ALTER INDEX. When you add a stopword dynamically, you need not re-index, because the word immediately becomes a stopword and is removed from the index.

Note: Even though you can dynamically add stopwords to an index, you cannot dynamically remove stopwords. To remove a stopword, you must use CTX_DDL.REMOVE_STOPWORD, drop your index and re-create it.

See Also: "ALTER INDEX" in Chapter 1, "Oracle Text SQL Statements and Operators"

System-Defined Preferences

When you install Oracle Text, some indexing preferences are created. You can use these preferences in the parameter clause of CREATE INDEX or define your own.

The default index parameters are mapped to some of the system-defined preferences described in this section.

See Also: For more information about default index parameters, see "Default Index Parameters" on page 2-77

System-defined preferences are divided into the following categories:

- Data Storage
- Filter

- Lexer
- Section Group
- Stoplist
- Storage
- Wordlist

Data Storage

This section discusses the types associated with data storage preferences.

CTXSYS.DEFAULT_DATASTORE

This preference uses the DIRECT_DATASTORE type. Use this preference to create indexes for text columns in which the text is stored directly in the column.

CTXSYS.FILE_DATASTORE

This preference uses the FILE_DATASTORE type.

CTXSYS.URL_DATASTORE

This preference uses the URL_DATASTORE type.

Filter

This section discusses the types associated with filtering preferences.

CTXSYS.NULL_FILTER

This preference uses the NULL_FILTER type.

CTXSYS.AUTO_FILTER

This preference uses the AUTO_FILTER type.

Lexer

This section discusses the types associated with lexer preferences.

CTXSYS.DEFAULT_LEXER

The default lexer depends on the language used at install time. The following sections describe the default settings for `CTXSYS.DEFAULT_LEXER` for each language.

American and English Language Settings If your language is English, this preference uses the BASIC_LEXER with the `index_themes` attribute disabled.

Danish Language Settings If your language is Danish, this preference uses the BASIC_LEXER with the following option enabled:

- Alternate spelling (alternate_spelling attribute set to `DANISH`)

Dutch Language Settings If your language is Dutch, this preference uses the BASIC_LEXER with the following options enabled:

- composite indexing (`composite` attribute set to `DUTCH`)

German and German DIN Language Settings If your language is German, then this preference uses the BASIC_LEXER with the following options enabled:

- Case-sensitive indexing (`mixed_case` attribute enabled)
- Composite indexing (`composite` attribute set to `GERMAN`)
- Alternate spelling (`alternate_spelling` attribute set to `GERMAN`)

Finnish, Norwegian, and Swedish Language Settings If your language is Finnish, Norwegian, or Swedish, this preference uses the BASIC_LEXER with the following option enabled:

- Alternate spelling (alternate_spelling attribute set to `SWEDISH`)

Japanese Language Settings If you language is Japanese, this preference uses the JAPANESE_VGRAM_LEXER.

Korean Language Settings If your language is Korean, this preference uses the KOREAN_MORPH_LEXER. All attributes for the `KOREAN_MORPH_LEXER` are enabled.

Chinese Language Settings If your language is Simplified or Traditional Chinese, this preference uses the CHINESE_VGRAM_LEXER.

Other Languages For all other languages not listed in this section, this preference uses the BASIC_LEXER with no attributes set.

> **See Also:** To learn more about these options, see "BASIC_LEXER" on page 2-31

CTXSYS.BASIC_LEXER

This preference uses the `BASIC_LEXER`.

Section Group

This section discusses the types associated with section group preferences.

CTXSYS.NULL_SECTION_GROUP

This preference uses the `NULL_SECTION_GROUP` type.

CTXSYS.HTML_SECTION_GROUP

This preference uses the `HTML_SECTION_GROUP` type.

CTXSYS.AUTO_SECTION_GROUP

This preference uses the `AUTO_SECTION_GROUP` type.

CTXSYS.PATH_SECTION_GROUP

This preference uses the `PATH_SECTION_GROUP` type.

Stoplist

This section discusses the types associated with stoplist preferences.

CTXSYS.DEFAULT_STOPLIST

This stoplist preference defaults to the stoplist of your database language.

> **See Also:** For a complete list of the stop words in the supplied stoplists, see Appendix E, "Oracle Text Supplied Stoplists"

CTXSYS.EMPTY_STOPLIST

This stoplist has no words.

Storage

This section discusses the types associated with storage preferences.

CTXSYS.DEFAULT_STORAGE

This storage preference uses the BASIC_STORAGE type.

Wordlist

This section discusses the types associated with wordlist preferences.

CTXSYS.DEFAULT_WORDLIST

This preference uses the language stemmer for your database language. If your language is not listed in Table 2–33 on page 2-58, then this preference defaults to the NULL stemmer and the GENERIC fuzzy matching attribute.

System Parameters

This section describes the Oracle Text system parameters, which are divided into the following categories:

- General System Parameters
- Default Index Parameters

General System Parameters

When you install Oracle Text, in addition to the system-defined preferences, the following system parameters are set:

Table 2–40 General System Parameters

System Parameter	Description
MAX_INDEX_MEMORY	This is the maximum indexing memory that can be specified in the parameter clause of CREATE INDEX and ALTER INDEX. The maximum value for this parameter is 2 GB -1.
DEFAULT_INDEX_MEMORY	This is the default indexing memory used with CREATE INDEX and ALTER INDEX.
LOG_DIRECTORY	This is the directory for CTX_OUTPUT log files.
CTX_DOC_KEY_TYPE	This is the default input key type, either ROWID or PRIMARY_KEY, for the CTX_DOC procedures. Set to ROWID at install time. **See Also:** CTX_DOC.SET_KEY_TYPE on page 8-35.

View system defaults by querying the CTX_PARAMETERS view. Change defaults using the CTX_ADM.SET_PARAMETER procedure.

Default Index Parameters

This section describes the index parameters that you can use when you create CONTEXT and CTXCAT indexes.

CONTEXT Index Parameters

The following default parameters are used when you create a CONTEXT index and do not specify preferences in the parameter clause of CREATE INDEX. Each default parameter names a system-defined preference to use for data storage, filtering, lexing, and so on.

Table 2–41 Default CONTEXT Index Parameters

Parameter	Used When	Default Value
DEFAULT_DATASTORE	No datastore preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_DATASTORE
DEFAULT_FILTER_FILE	No filter preference specified in parameter clause of CREATE INDEX, and either of the following conditions is true: ■ Your files are stored in external files (BFILES) or ■ Specify a datastore preference that uses FILE_DATASTORE	CTXSYS.AUTO_FILTER
DEFAULT_FILTER_BINARY	No filter preference specified in parameter clause of CREATE INDEX, and Oracle Text detects that the text column datatype is RAW, LONG RAW, or BLOB.	CTXSYS.AUTO_FILTER
DEFAULT_FILTER_TEXT	No filter preference specified in parameter clause of CREATE INDEX, and Oracle Text detects that the text column datatype is either LONG, VARCHAR2, VARCHAR, CHAR, or CLOB.	CTXSYS.NULL_FILTER
DEFAULT_SECTION_HTML	No section group specified in parameter clause of CREATE INDEX, and when either of the following conditions is true: ■ Your datastore preference uses URL_DATASTORE or ■ Your filter preference uses AUTO_FILTER.	CTXSYS.HTML_SECTION_GROUP
DEFAULT_SECTION_TEXT	No section group specified in parameter clause of CREATE INDEX, and when you do *not* use either URL_DATASTORE or AUTO_FILTER.	CTXSYS.NULL_SECTION_GROUP

Table 2–41 (Cont.) Default CONTEXT Index Parameters

Parameter	Used When	Default Value
DEFAULT_STORAGE	No storage preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STORAGE
DEFAULT_LEXER	No lexer preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_LEXER
DEFAULT_STOPLIST	No stoplist specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STOPLIST
DEFAULT_WORDLIST	No wordlist preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_WORDLIST

CTXCAT Index Parameters

The following default parameters are used when you create a CTXCAT index with CREATE INDEX and do not specify any parameters in the parameter string. The CTXCAT index supports only the index set, lexer, storage, stoplist, and wordlist parameters. Each default parameter names a system-defined preference.

Table 2–42 Default CTXCAT Index Parameters

Parameter	Used When	Default Value
DEFAULT_CTXCAT_INDEX_SET	No index set specified in parameter clause of CREATE INDEX.	n/a
DEFAULT_CTXCAT_STORAGE	No storage preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STORAGE
DEFAULT_CTXCAT_LEXER	No lexer preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_LEXER
DEFAULT_CTXCAT_STOPLIST	No stoplist specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STOPLIST
DEFAULT_CTXCAT_WORDLIST	No wordlist preference specified in parameter clause of CREATE INDEX. Note that while you can specify a wordlist preference for CTXCAT indexes, most of the attributes do not apply, because the catsearch query language does not support wildcarding, fuzzy, and stemming. The only attribute that is useful is PREFIX_INDEX for Japanese data.	CTXSYS.DEFAULT_WORDLIST

CTXRULE Index Parameters

Table 2–43 lists the default parameters that are used when you create a CTXRULE index with CREATE INDEX and do not specify any parameters in the parameter string. The

CTXRULE index supports only the lexer, storage, stoplist, and wordlist parameters. Each default parameter names a system-defined preference.

Table 2–43 Default CTXRULE Index Parameters

Parameter	Used When	Default Value
DEFAULT_CTXRULE_LEXER	No lexer preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_LEXER
DEFAULT_CTXRULE_STORAGE	No storage preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STORAGE
DEFAULT_CTXRULE_STOPLIST	No stoplist specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STOPLIST
DEFAULT_CTXRULE_WORDLIST	No wordlist preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_WORDLIST
DEFAULT_CLASSIFIER	No classifier preference is specified in parameter clause.	RULE_CLASSIFIER

CTXRULE Index Limitations

The CTXRULE index does not support the following query operators:

- Fuzzy
- Soundex

It also does not support the following BASIC_WORDLIST attributes:

- SUBSTRING_INDEX
- PREFIX_INDEX

Viewing Default Values

View system defaults by querying the CTX_PARAMETERS view. For example, to see all parameters and values, enter the following statement:

```
SQL> SELECT par_name, par_value from ctx_parameters;
```

Changing Default Values

Change a default value using the CTX_ADM.SET_PARAMETER procedure to name another custom or system-defined preference to use as default.

3

Oracle Text CONTAINS Query Operators

This chapter describes operator precedence and provides descriptions, syntax, and examples for every CONTAINS operator. The following topics are covered:

- Operator Precedence
- ABOUT
- ACCUMulate (,)
- AND (&)
- Broader Term (BT, BTG, BTP, BTI)
- DEFINEMERGE
- DEFINESCORE
- EQUIValence (=)
- Fuzzy
- HASPATH
- INPATH
- MDATA
- MINUS (-)
- MNOT
- Narrower Term (NT, NTG, NTP, NTI)
- NDATA
- NEAR (;)
- NOT (~)
- OR (|)
- Preferred Term (PT)
- Related Term (RT)
- SDATA
- soundex (!)
- stem ($)
- Stored Query Expression (SQE)
- SYNonym (SYN)

- threshold (>)
- Translation Term (TR)
- Translation Term Synonym (TRSYN)
- Top Term (TT)
- weight (*)
- wildcards (% _)
- WITHIN

Operator Precedence

Operator precedence determines the order in which the components of a query expression are evaluated. Text query operators can be divided into two sets of operators that have their own order of evaluation. These two groups are described later as Group 1 and Group 2.

In all cases, query expressions are evaluated in order from left to right according to the precedence of their operators. Operators with higher precedence are applied first. Operators of equal precedence are applied in order of their appearance in the expression from left to right.

Group 1 Operators

Within query expressions, the Group 1 operators have the following order of evaluation from highest precedence to lowest:

1. EQUIValence (=)
2. NEAR (;)
3. weight (*), threshold (>)
4. MINUS (-)
5. NOT (~)
6. MNOT
7. WITHIN
8. AND (&)
9. OR (|)
10. ACCUMulate (,)

Group 2 Operators and Characters

Within query expressions, the Group 2 operators have the following order of evaluation from highest to lowest:

1. Wildcard Characters
2. stem ($)
3. Fuzzy
4. soundex (!)

Procedural Operators

Other operators not listed under Group 1 or Group 2 are procedural. These operators have no sense of precedence attached to them. They include the SQE and thesaurus operators.

Precedence Examples

Table 3–1 Query Expression Precedence Examples

Query Expression	Order of Evaluation
w1 | w2 & w3	(w1) | (w2 & w3)
w1 & w2 | w3	(w1 & w2) | w3
?w1, w2 | w3 & w4	(?w1), (w2 | (w3 & w4))
abc = def ghi & jkl = mno	((abc = def) ghi) & (jkl=mno)
dog and cat WITHIN body	dog and (cat WITHIN body)

In the first example, because `AND` has a higher precedence than `OR`, the query returns all documents that contain *w1* and all documents that contain both *w2* and *w3*.

In the second example, the query returns all documents that contain both *w1* and *w2* and all documents that contain *w3*.

In the third example, the fuzzy operator is first applied to *w1*, then the `AND` operator is applied to arguments *w3* and *w4*, then the `OR` operator is applied to term *w2* and the results of the `AND` operation, and finally, the score from the fuzzy operation on *w1* is added to the score from the `OR` operation.

The fourth example shows that the equivalence operator has higher precedence than the `AND` operator.

The fifth example shows that the `AND` operator has lower precedence than the `WITHIN` operator.

Altering Precedence

Precedence is altered by grouping characters as follows:

- Within parentheses, expansion or execution of operations is resolved before other expansions regardless of operator precedence.
- Within parentheses, precedence of operators is maintained during evaluation of expressions.
- Within parentheses, expansion operators are not applied to expressions unless the operators are also within the parentheses.

> **See Also:** "Grouping Characters" in Chapter 4, "Special Characters in Oracle Text Queries"

ABOUT

General Behavior

Use the ABOUT operator to return documents that are related to a query term or phrase. In English and French, ABOUT enables you to query on concepts, even if a concept is not actually part of a query. For example, an ABOUT query on *heat* might return documents related to temperature, even though the term *temperature* is not part of the query.

In other languages, using ABOUT will often increase the number of returned documents and may improve the sorting order of results. For all languages, Oracle Text scores results for an ABOUT query with the most relevant document receiving the highest score.

English and French Behavior

In English and French, use the ABOUT operator to query on concepts. The system looks up concept information in the theme component of the index. Create a theme component to your index by setting the INDEX_THEMES BASIC_LEXER attribute to YES.

Note: You need not have a theme component in the index to enter ABOUT queries in English and French. However, having a theme component in the index yields the best results for ABOUT queries.

Oracle Text retrieves documents that contain concepts that are related to your query word or phrase. For example, if you enter an ABOUT query on *California*, the system might return documents that contain the terms *Los Angeles* and *San Francisco*, which are cities in California.The document need not contain the term *California* to be returned in this ABOUT query.

The word or phrase specified in your ABOUT query need not exactly match the themes stored in the index. Oracle Text normalizes the word or phrase before performing lookup in the index.

You can use the ABOUT operator with the CONTAINS and CATSEARCH SQL operators. In the case of CATSEARCH, you must use query templating with the CONTEXT grammar to query on the indexed themes. See ABOUT Query with CATSEARCH in the Examples section.

Syntax

Syntax	Description
about(*phrase*)	In all languages, increases the number of relevant documents returned for the same query without the ABOUT operator.The *phrase* parameter can be a single word or a phrase, or a string of words in free text format. In English and French, returns documents that contain concepts related to *phrase*, provided the BASIC_LEXER INDEX_THEMES attribute is set to YES at index time. The score returned is a relevance score. Oracle Text ignores any query operators that are included in *phrase*. If your index contains only theme information, an ABOUT operator and operand must be included in your query on the text column or else Oracle Text returns an error. The *phrase* you specify cannot be more than 4000 characters.

Case-Sensitivity

ABOUT queries give the best results when your query is formulated with proper case. This is because the normalization of your query is based on the knowledge catalog which is case-sensitive.

However, you need not type your query in exact case to obtain results from an ABOUT query. The system does its best to interpret your query. For example, if you enter a query of *CISCO* and the system does not find this in the knowledge catalog, the system might use *Cisco* as a related concept for look-up.

Improving ABOUT Results

The ABOUT operator uses the supplied knowledge base in English and French to interpret the phrase you enter. Your ABOUT query therefore is limited to knowing and interpreting the concepts in the knowledge base.

Improve the results of your ABOUT queries by adding your application-specific terminology to the knowledge base.

> **See Also:** "Extending the Knowledge Base" in Chapter 14, "Oracle Text Utilities"

Limitations

The phrase you specify in an ABOUT query cannot be more than 4000 characters.

Examples

Single Words

To search for documents that are about soccer, use the following syntax:

```
'about(soccer)'
```

Phrases

Further refine the query to include documents about soccer rules in international competition by entering the phrase as the query term:

```
'about(soccer rules in international competition)'
```

In this English example, Oracle Text returns all documents that have themes of *soccer*, *rules*, or *international competition*.

In terms of scoring, documents which have all three themes will generally score higher than documents that have only one or two of the themes.

Unstructured Phrases

You can also query on unstructured phrases, such as the following:

```
'about(japanese banking investments in indonesia)'
```

Combined Queries

Use other operators, such as AND or NOT, to combine ABOUT queries with word queries. For example, enter the following combined ABOUT and word query:

```
'about(dogs) and cat'
```

Combine an ABOUT query with another ABOUT query as follows:

```
'about(dogs) not about(labradors)'
```

> **Note:** You cannot combine ABOUT with the WITHIN operator, as for example '*ABOUT (xyz) WITHIN abc'*.

ABOUT Query with CATSEARCH

Enter ABOUT queries with CATSEARCH using the query template method with grammar set to CONTEXT as follows:

```
select pk||' ==> '||text from test
where catsearch(text,
'<query>
  <textquery grammar="context">
     about(California)
  </textquery>
  <score datatype="integer"/>
</query>','')>0
order by pk;
```

ACCUMulate (,)

Use the ACCUM operator to search for documents that contain at least one occurrence of any query terms, with the returned documents ranked by a cumulative score based on how many query terms are found (and how frequently).

Syntax

Syntax	Description
term1,*term2* *term1* ACCUM *term2*	Returns documents that contain *term1* or *term2*. Ranks documents according to document term weight, with the highest scores assigned to documents that have the highest total term weight.

ACCUMulate Scoring

ACCUMulate first scores documents on how many query terms a document matches. A document that matches more terms will always score higher than a document that matches fewer terms, even if the terms appear more frequently in the latter. In other words, if you search for *dog ACCUM cat*, you'll find that

```
the dog played with the cat
```

scores higher than

```
the big dog played with the little dog while a third dog ate the dog food
```

Scores are divided into ranges. In a two-term ACCUM, hits that match both terms will always score between 51 and 100, whereas hits matching only one of the terms will score between 1 and 50. Likewise, for a three-term ACCUM, a hit matching one term will score between 1 and 33; a hit matching two terms will score between 34 and 66, and a hit matching all three terms will score between 67 and 100. *Within these ranges*, normal scoring algorithms apply.

> **See Also:** Appendix F, "The Oracle Text Scoring Algorithm" for more information on how scores are calculated

You can assign different weights to different terms. For example, in a query of the form

```
soccer, Brazil*3
```

the term *Brazil* is weighted three times as heavily as *soccer*. Therefore, the document

```
people play soccer because soccer is challenging and fun
```

will score lower than

```
Brazil is the largest nation in South America
```

but both documents will rank below

```
soccer is the national sport of Brazil
```

Note that a query of *soccer ACCUM Brazil*3* is equivalent to *soccer ACCUM Brazil ACCUM Brazil ACCUM Brazil*. Because each query term *Brazil* is considered independent, the entire query is scored as though it has four terms, not two, and thus has four scoring ranges. The first Brazil-and-soccer example document shown above

scores in the first range (1-25), the second scores in the third range (51-75), and the third scores in the fourth range (76-100). (No document scores in the second range, because any document with *Brazil* in it will be considered to match at least three query terms.)

Example

```
set serveroutput on;
DROP TABLE accumtbl;
CREATE TABLE accumtbl (id NUMBER, text VARCHAR2(4000) );

INSERT INTO accumtbl VALUES ( 1, 'the little dog played with the big dog
      while the other dog ate the dog food');
INSERT INTO accumtbl values (2, 'the cat played with the dog');

CREATE INDEX accumtbl_idx ON accumtbl (text) indextype is ctxsys.context;

PROMPT dog ACCUM cat
SELECT SCORE(10) FROM accumtbl WHERE CONTAINS (text, 'dog ACCUM cat', 10)
   > 0;

PROMPT dog*3 ACCUM cat
SELECT SCORE(10) FROM accumtbl WHERE CONTAINS (text, 'dog*3 ACCUM cat', 10)
   > 0;
```

This produces the following output. Note that the document with both *dog* and *cat* scores highest.

```
dog ACCUM cat
   ID  SCORE(10)
----- ----------
    1          6
    2         52

dog*3 ACCUM cat
   ID  SCORE(10)
----- ----------
    1         53
    2         76
```

Related Topics

See also weight (*) on page 3-56

AND (&)

Use the AND operator to search for documents that contain at least one occurrence of each of the query terms.

Syntax

Syntax	Description
term1&*term2* *term1* and *term2*	Returns documents that contain *term1* and *term2*. Returns the minimum score of its operands. All query terms must occur; lower score taken.

Example

To obtain all the documents that contain the terms *blue* and *black* and *red*, enter the following query:

```
'blue & black & red'
```

In an AND query, the score returned is the score of the lowest query term. In this example, if the three individual scores for the terms *blue*, *black*, and *red* is 10, 20 and 30 within a document, the document scores 10.

Related Topics

See Also: The AND operator returns documents that contain *all* of the query terms, while OR operator returns documents that contain *any* of the query terms. See "OR (|)" on page 3-41.

Broader Term (BT, BTG, BTP, BTI)

Use the broader term operators (BT, BTG, BTP, BTI) to expand a query to include the term that has been defined in a thesaurus as the broader or higher level term for a specified term. They can also expand the query to include the broader term for the broader term and the broader term for that broader term, and so on up through the thesaurus hierarchy.

Syntax

Syntax	Description
BT(*term*[(*qualifier*)][,*level*][,*thes*])	Expands a query to include the term defined in the thesaurus as a broader term for term.
BTG(*term*[(*qualifier*)][,*level*][,*thes*])	Expands a query to include all terms defined in the thesaurus as broader generic terms for term.
BTP(*term*[(*qualifier*)][,*level*][,*thes*])	Expands a query to include all the terms defined in the thesaurus as broader partitive terms for term.
BTI(*term*[(*qualifier*)][,*level*][,*thes*])	Expands a query to include all the terms defined in the thesaurus as broader instance terms for term.

term
Specify the operand for the broader term operator. Oracle Text expands term to include the broader term entries defined for the term in the thesaurus specified by thes. For example, if you specify *BTG(dog)*, the expansion includes only those terms that are defined as broader term generic for *dog*. You cannot specify expansion operators in the term argument.

The number of broader terms included in the expansion is determined by the value for level.

qualifier
Specify a qualifier for term, if term is a homograph (word or phrase with multiple meanings, but the same spelling) that appears in two or more nodes in the same hierarchy branch of thes.

If a qualifier is not specified for a homograph in a broader term query, the query expands to include the broader terms of all the homographic terms.

level
Specify the number of levels traversed in the thesaurus hierarchy to return the broader terms for the specified term. For example, a level of 1 in a BT query returns the broader term entry, if one exists, for the specified term. A level of 2 returns the broader term entry for the specified term, as well as the broader term entry, if one exists, for the broader term.

The level argument is optional and has a default value of one (1). Zero or negative values for the level argument return only the original query term.

thes
Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. A thesaurus named DEFAULT *must* exist in the thesaurus tables if you use this default value.

Note: If you specify `thes`, then you must also specify `level`.

Examples

The following query returns all documents that contain the term *tutorial* or the `BT` term defined for *tutorial* in the `DEFAULT` thesaurus:

```
'BT(tutorial)'
```

When you specify a thesaurus name, you must also specify `level` as in:

```
'BT(tutorial, 2, mythes)'
```

Broader Term Operator on Homographs

If *machine* is a broader term for *crane (building equipment)* and *bird* is a broader term for *crane (waterfowl)* and no qualifier is specified for a broader term query, the query

```
BT(crane)
```

expands to:

```
'{crane} or {machine} or {bird}'
```

If *waterfowl* is specified as a qualifier for *crane* in a broader term query, the query

```
BT(crane{(waterfowl)})
```

expands to the query:

```
'{crane} or {bird}'
```

Note: When specifying a qualifier in a broader or narrower term query, the qualifier and its notation (parentheses) must be escaped, as is shown in this example.

Related Topics

Browse a thesaurus using procedures in the `CTX_THES` package.

See Also: CTX_THES.BT in Chapter 12, "CTX_THES Package" for more information on browsing the broader terms in your thesaurus

DEFINEMERGE

Use the DEFINEMERGE operator to define how the score of child nodes of the AND and OR should be merged. The DEFINEMERGE operator can be used as operand(s) of any operators that allow AND or OR as operands. The score can be merged in three ways: picking the minimum value, picking the maximum value, or calculating the average score of all child nodes.

Use DEFINESCORE before using DEFINEMERGE.

Syntax

```
DEFINEMERGE ( (  (text_query1), (text_query2), … )  , operator, merge_method )
```

Syntax	Description
text_query1,2 ...	Defines the search criteria. These parameters can have any value that is valid for the AND/OR operator.
operator	Defines the relationship between the two text_query parameters.
merge_method	Defines how the score of the text_query should be merged. Possible values: MIN, MAX, AVG, ADD

Example

Example 3–1 DEFINEMERGE and text_query

The following examples show only the text_query part of a CONTAINS query:

```
'DEFINEMERGE ( ((dog), (cat)), OR, AVG)'
```

Queries for the term "dog" or "cat," using the average relevance score of both terms as the merged score.

```
'DEFINEMERGE (((dog , cat) , (blue or black)), AND, MIN )'
```

Queries for the expression "dog ACCUM cat" and "blue OR black," using the default scoring schemes and then using the minimum score of the two as the merged-score.

```
'DEFINEMERGE( ((DEFINESCORE(dog, DISCRETE)) , (cat)), AND, MAX)'
```

Queries for the term "dog" using the DISCRETE scoring, and for the term "cat" using the default relevant scoring, and then using the maximum score of the two as the merged-score.

Related Topic

DEFINESCORE on page 3-13.

DEFINESCORE

Use the DEFINESCORE operator to define how a term or phrase, or a set of term equivalences will be scored. The definition of a scoring expression can consist of an arithmetic expression of predefined scoring components and numeric literals.

DEFINEMERGE can be used after DEFINESCORE.

Syntax

```
DEFINESCORE (query_term, scoring_expression)
```

query_term
The query term or phrase. Expressions containing the following operators are also allowed:

-	-
ABOUT	EQUIV(=)
Fuzzy	Soundex (!)
Stem ($)	Wildcards (% _)
SDATA	MDATA

scoring_expression
An arithmetic expression that describes how the query_term should be scored. This operand is a string that contains the following components:

- Arithmetic operators: + - * /. The precedence is multiplication and division (*, /) first before addition and subtraction (+, -).
- Grouping operators: (). Parentheses can be used to alter the precedence of the arithmetic operators.
- Absolute function: ABS(n) returns the absolute value of n; where n is any expression that returns a number.
- Logarithmic function: LOG(n) returns the base-10 logarithmic value of n ; where n is any expression that returns a number.
- Predefined scoring components: Each of the following scoring components returns a value of 0 - 100, depending on different criteria:

Name	Description
DISCRETE	If the term exists in the document, score = 100. Otherwise, score = 0.
OCCURRENCE	Score based on the number of occurrences.
RELEVANCE	Score based on the document's relevance.
COMPLETION	Score based on coverage. Documents will score higher if the ratio between the number of the matching terms and the number of all terms in the section (counting stop words) is higher. The COMPLETION scoring is only applicable when used with the WITHIN operator to search in zone sections.

Name	Description
IGNORE	Ignore the scoring of this term. This component should be used alone. Otherwise, the query will return a syntax error. If the scoring of the only term in the query is set to IGNORE, then all the matching documents should be returned with the same score of 100.

Note: For numeric literals, any number literal can be used that conforms to the SQL pattern of number literal, and is within the range of the double precision floating point (-3.4e38 to 3.4e38).

scoring_expression Syntax

```
<Exp>   :=         <Exp> + <Term> | <Exp> - <Term> |  <Term>

<Term>   :=         <Term> * <Factor> | <Term> / <Factor> | <Factor>

<Factor> :=         <<NumericLiterals >>| DISCRETE | OCCURRENCE | RELEVANCE |
         COMPLETION | IGNORE |  ( <Exp> ) | -<Factor> | Abs(<Exp>) | Log(<Exp>)
```

Examples

```
'DEFINESCORE (dog, OCCURRENCE)'
```
Queries for the word *dog*, and scores each document using the occurrence score. Returns the score as integer.

```
'DEFINESCORE (Labradors are big dog, RELEVANCE)'
```
Queries for the phrase *Labradors are big dogs*, and scores each document using the relevance score.

```
'cat and DEFINESCORE (dog, IGNORE)'
```
Queries for the words *dog* and *cat*, using only the default relevance score of *cat* as the overall score of the document. Returns the score as integer.

```
'DEFINESCORE (dog, IGNORE)'
```
Queries for the word *dog*, and returns all documents with the word *dog*. The result is the same as if all documents get a score of 100. Returns the score as integer.

```
'DEFINESCORE (dog, ABS (100-RELEVANCE))'
```
Queries for the word *dog*, and scores each document using the absolute value of 100 minus the relevance score. Returns the score as integer.

```
'cat and DEFINESCORE (dog, RELEVANCE*5 - OCCURRENCE)'
```
Returns a syntax error: Two pre-defined components are used.

When DEFINESCORE is used with query templates, the scoring_expression overrides the values specified by the template. The following example queries for "dog" and "cat," scores "cat" using OCCURRENCE(COUNT) and scores "dog" based on RELEVANCE.

```
query>
      <textquery grammar="CONTEXT" lang="english">
                DEFINESCORE(dog, RELEVANCE) and  cat
      </textquery>
      <score datatype="INTEGER" algorithm="COUNT"/>
</query>
```

Limitations

- If the ABOUT operator is used in query_term, the OCCURRENCE and COMPLETION scoring will not be applicable. If used, the query will return a syntax error.
- The IGNORE score cannot be used as right hand of the minus operator. If used, then a syntax error will occur.
- The COMPLETION score is only applicable if the DEFINESCORE is used with a WITHIN operator to search in zone sections, for example:

  ```
  'DEFINESCORE (dog, COMPLETION) within zonesection'
  ```
  otherwise, the query will return a syntax error.
- For the left hand operand of WITHIN:
  - All nodes must use the same predefined-scoring component. (If not specified, then the predefined scoring is RELEVANCE.)
  - If the nodes use DISCRETE or COMPLETION, then only the AND and OR operator is allowed as the left hand children of WITHIN.
  - If the nodes use DISCRETE or COMPLETION, then WITHIN will use the max score of all section instances as the score.
  - If the nodes use RELEVANCE or OCCURRENCE, then WITHIN will use the summation of the score of all section instances as the score.
- Only one predefined scoring component can be used in the scoring_expression at one time. If more than one predefined scoring component is used, then a syntax error will occur.

See Also: *Oracle Database SQL Language Reference*

Notes

- The DEFINESCORE operator, the absolute function, the logarithmic function, and the pre-defined scoring components are case-insensitive.
- The query_term and the scoring_expression parameters are mandatory.
- The final score of the DEFINESCORE operator will be truncated to be in the 0 – 100 range. If the data type is INTEGER, then the score is rounded up.
- The intermediate data type of the scoring value is a double precision float. As a result, the value is limited to be in the -3.4e38 to 3.4e38 range. If the intermediate scoring of any document exceeds the value, then the score will be truncated. If an integer scoring is required, then the score will always be rounded up after the score is calculated.
- The DEFINESCORE operator can be used as an operand of the following operators:
  - AND
  - NOT
  - INPATH
  - THRESHOLD
  - WITHIN
  - SQE
  - OR

  - DEFINEMERGE
  - MINUS
  - WEIGHT
  - ACCUM

  For example, the following statement is valid:

  ```
  DEFINESCORE('dog', OCCURRENCE) AND DEFINESCORE('cat', RELEVANCE)
  ```

  Queries for the term "dog" using occurrence scoring, and the term "cat" using relevance scoring.

- If `DEFINESCORE` is used as a parameter of other operators, then an error will be returned. For example, the following example returns an error:

  ```
  SYN(DEFINESCORE('cat', OCCURRENCE))
  ```

- When used with query templates, the `scoring_expression` overrides the values specified by the template. For example,

  ```
  query>
      <textquery grammar="CONTEXT" lang="english">
          DEFINESCORE(dog, RELEVANCE) and  cat
      </textquery>
      <score datatype="INTEGER" algorithm="COUNT"/>
  </query>
  ```

  Queries for "dog" and "cat", scores "cat" using `OCCURRENCE(COUNT)`, and scores "dog" based on `RELEVANCE`.

Related Topic

DEFINEMERGE on page 3-12.

EQUIValence (=)

Use the EQUIV operator to specify an acceptable substitution for a word in a query.

Syntax

Syntax	Description
term1=*term2* *term1* equiv *term2*	Specifies that term2 is an acceptable substitution for term1. Score calculated as the sum of all occurrences of both terms.

Example

The following example returns all documents that contain either the phrase *alsatians are big dogs* or *labradors are big dogs*:

```
'labradors=alsatians are big dogs'
```

Operator Precedence

The EQUIV operator has higher precedence than all other operators except the expansion operators (fuzzy, soundex, stem).

Fuzzy

Use the `fuzzy` operator to expand queries to include words that are spelled similarly to the specified term. This type of expansion is helpful for finding more accurate results when there are frequent misspellings in your document set.

The `fuzzy` syntax enables you to rank the result set so that documents that contain words with high similarity to the query word are scored higher than documents with lower similarity. You can also limit the number of expanded terms.

Unlike stem expansion, the number of words generated by a `fuzzy` expansion depends on what is in the index. Results can vary significantly according to the contents of the index.

Supported Languages

Oracle Text supports `fuzzy` definitions for English, French, German, Italian, Dutch, Spanish, Portuguese, Japanese, OCR, and auto-language detection.

Stopwords

If the `fuzzy` expansion returns a stopword, the stopword is not included in the query or highlighted by `CTX_DOC.HIGHLIGHT` or `CTX_DOC.MARKUP`.

Base-Letter Conversion

If base-letter conversion is enabled for a text column and the query expression contains a `fuzzy` operator, Oracle Text operates on the base-letter form of the query.

Syntax

```
fuzzy(term, score, numresults, weight)
```

Parameter	Description
term	Specify the word on which to perform the `fuzzy` expansion. Oracle Text expands term to include words only in the index. The word needs to be at least 3 characters for the `fuzzy` operator to process it.
score	Specify a similarity score. Terms in the expansion that score below this number are discarded. Use a number between 1 and 80. The default is 60.
numresults	Specify the maximum number of terms to use in the expansion of term. Use a number between 1 and 5000. The default is 100.
weight	Specify `WEIGHT` or `W` for the results to be weighted according to their similarity scores. Specify `NOWEIGHT` or `N` for no weighting of results.

Examples

Consider the `CONTAINS` query:

```
...CONTAINS(TEXT, 'fuzzy(government, 70, 6, weight)', 1) > 0;
```

This query expands to the first six `fuzzy` variations of *government* in the index that have a similarity score over 70.

In addition, documents in the result set are weighted according to their similarity to *government*. Documents containing words most similar to government receive the highest score.

Skip unnecessary parameters using the appropriate number of commas. For example:

```
'fuzzy(government,,,weight)'
```

Backward Compatibility Syntax

The old `fuzzy` syntax from previous releases is still supported. This syntax is as follows:

Parameter	Description
?term	Expands term to include all terms with similar spellings as the specified term. Term needs to be at least 3 characters for the `fuzzy` operator to process it.

HASPATH

Use this operator to find all XML documents that contain a specified section path. You can also use this operator to do section equality testing.

Your index must be created with the PATH_SECTION_GROUP for this operator to work.

Syntax

Syntax	Description
HASPATH(path)	Searches an XML document set and returns a score of 100 for all documents where *path* exists. Separate parent and child paths with the / character. For example, you can specify *A/B/C*. See example.
HASPATH(A="value")	Searches an XML document set and returns a score of 100 for all documents that have the element A with content *value* and only *value*. See example.

Using Special Characters with HASPATH and INPATH

The following rules govern the use of special characters with regard to both the HASPATH and INPATH operators:

- Left-brace ({) and right-brace (}) characters are not allowed inside HASPATH or INPATH expressions unless they are inside the equality operand enclosed by double quotes. So both 'HASPATH({/A/B})' and 'HASPATH(/A/{B})' will return errors. However, 'HASPATH(/A[B="{author}"])' will be parsed correctly.
- With exception of the backslash (\), special characters, such as dollar sign ($), percent sign (%), underscore (_), left brace ({), and right brace (}), when inside the equality operand enclosed by double or single quotes, have no special meaning. (That is, no stemming, wildcard expansion, or similar processing will be performed on them.) However, they are still subject to regular text lexing and will be translated to whitespace, with the exception of characters declared as printjoins. A backslash will still escape any character that immediately follows it.

  For example, if the hyphen (-) and the double quote character (") are defined as printjoins in a lexer preference, then:

  - The string *B_TEXT* inside HASPATH(/A[B="B_TEXT") will be lexed as the phrase *B TEXT*.
  - The string *B-TEXT* inside HASPATH(/A[B="B-TEXT") will be lexed as the word *B-TEXT*.
  - The string *B'TEXT* inside HASPATH(/A[B="B'TEXT") will be lexed as the word *B"TEXT*. You must use a backslash to escape the double quote between *B* and *TEXT*, or you will get a parsing error.
  - The string *{B_TEXT}* inside HASPATH(/A[B="{B_TEXT}") will be lexed as a phrase *B TEXT*.

Example

Path Testing

The query

```
HASPATH(A/B/C)
```

finds and returns a score of 100 for the document

```
<A><B><C>dog</C></B></A>
```

without the query having to reference *dog* at all.

Section Equality Testing

The query

```
dog INPATH A
```

finds

```
<A>dog</A>
```

but it also finds

```
<A>dog park</A>
```

To limit the query to the term *dog* and nothing else, you can use a section equality test with the `HASPATH` operator. For example,

```
HASPATH(A="dog")
```

finds and returns a score of 100 only for the first document, and not the second.

Limitations

Because of how XML section data is recorded, false matches might occur with XML sections that are completely empty as follows:

<A><B><C></C></B><D><E></E></D></A>

A query of `HASPATH(A/B/E)` or `HASPATH(A/D/C)` falsely matches this document. This type of false matching can be avoided by inserting text between empty tags.

INPATH

Use this operator to do path searching in XML documents. This operator is like the `WITHIN` operator except that the right-hand side is a parentheses enclosed path, rather than a single section name.

Your index must be created with the `PATH_SECTION_GROUP` for the `INPATH` operator to work.

Syntax

The `INPATH` operator has the following syntax:

Top-Level Tag Searching

Syntax	Description
term INPATH (/A) term INPATH (A)	Returns documents that have *term* within the <A> and </A> tags.

Any-Level Tag Searching

Syntax	Description
term INPATH (//A)	Returns documents that have *term* in the <A> tag at any level. This query is the same as *'term WITHIN A*'

Direct Parentage Path Searching

Syntax	Description
term INPATH (A/B)	Returns documents where *term* appears in a B element which is a direct child of a top-level A element. For example, a document containing `<A><B>term</B></A>` is returned.

Single-Level Wildcard Searching

Syntax	Description
term INPATH (A/*/B)	Returns documents where *term* appears in a B element which is a grandchild (two levels down) of a top-level A element. For example a document containing `<A><D><B>term</B></D></A>` is returned.

Multi-level Wildcard Searching

Syntax	Description
term INPATH (A/*/B/*/*/C)	Returns documents where *term* appears in a C element which is 3 levels down from a B element which is two levels down (grandchild) of a top-level A element.

Any-Level Descendant Searching

Syntax	Description
term INPATH(A//B)	Returns documents where term appears in a B element which is some descendant (any level) of a top-level A element.

Attribute Searching

Syntax	Description
term INPATH (//A/@B)	Returns documents where *term* appears in the B attribute of an A element at any level. Attributes must be bound to a direct parent.

Descendant/Attribute Existence Testing

Syntax	Description
term INPATH (A[B])	Returns documents where term appears in a top-level A element which has a B element as a direct child.
term INPATH (A[.//B])	Returns documents where term appears in a top-level A element which has a B element as a descendant at any level.
term INPATH (//A[@B])	Finds documents where term appears in an A element at any level which has a B attribute. Attributes must be tied to a direct parent.

Attribute Value Testing

Syntax	Description
term INPATH (A[@B = "value"])	Finds all documents where *term* appears in a top-level A element which has a B attribute whose value is *value*.
term INPATH (A[@B != "value"])	Finds all documents where *term* appears in a top-level A element which has a B attribute whose value is not *value*.

Tag Value Testing

Syntax	Description
term INPATH (A[B = "value"]))	Returns documents where *term* appears in an A tag which has a B tag whose value is *value*.

Not

Syntax	Description
term INPATH (A[NOT(B)])	Finds documents where *term* appears in a top-level A element which does not have a B element as an immediate child.

AND and OR Testing

Syntax	Description
term INPATH (A[B and C])	Finds documents where term appears in a top-level A element which has a B and a C element as an immediate child.
term INPATH (A[B and @C="value"]])	Finds documents where *term* appears in a top-level A element which has a B element and a C attribute whose value is *value*.
term INPATH (A [B OR C])	Finds documents where *term* appears in a top-level A element which has a B element or a C element.

Combining Path and Node Tests

Syntax	Description
term INPATH (A[@B = "value"]/C/D)	Returns documents where *term* appears in aD element which is the child of a C element, which is the child of a top-level A element with a B attribute whose value is *value*.

Nested INPATH

Nest the entire `INPATH` expression in another `INPATH` expression as follows:

```
(dog INPATH (//A/B/C)) INPATH (D)
```

When you do so, the two `INPATH` paths are completely independent. The outer `INPATH` path does not change the context node of the inner `INPATH` path. For example:

```
(dog INPATH (A)) INPATH (D)
```

never finds any documents, because the inner `INPATH` is looking for *dog* within the top-level tag A, and the outer `INPATH` constrains that to document with top-level tag D. A document can have only one top-level tag, so this expression never finds any documents.

Case-Sensitivity

Tags and attribute names in path searching are case-sensitive. That is,

```
dog INPATH (A)
```

finds `<A>dog</A>` but does not find `<a>dog</a>`. Instead use

```
dog INPATH (a)
```

Using Special Characters with INPATH

See "Using Special Characters with HASPATH and INPATH" on page 3-20 for information on using special characters, such as the percent sign (%) or the backslash (\), with `INPATH`.

Examples

Top-Level Tag Searching

To find all documents that contain the term *dog* in the top-level tag <A>:

```
dog INPATH (/A)
```

or

```
dog INPATH(A)
```

Any-Level Tag Searching

To find all documents that contain the term *dog* in the <A> tag at any level:

```
dog INPATH(//A)
```

This query finds the following documents:

```
<A>dog</A>
```

and

```
<C><B><A>dog</A></B></C>
```

Direct Parentage Searching

To find all documents that contain the term *dog* in a B element that is a direct child of a top-level A element:

```
dog INPATH(A/B)
```

This query finds the following XML document:

```
<A><B>My dog is friendly.</B><A>
```

but does not find:

```
<C><B>My dog is friendly.</B></C>
```

Tag Value Testing

You can test the value of tags. For example, the query:

```
dog INPATH(A[B="dog"])
```

Finds the following document:

```
<A><B>dog</B></A>
```

But does not find:

```
<A><B>My dog is friendly.</B></A>
```

Attribute Searching

You can search the content of attributes. For example, the query:

```
dog INPATH(//A/@B)
```

Finds the document

```
<C><A  B="snoop dog"> </A> </C>
```

Attribute Value Testing

You can test the value of attributes. For example, the query

```
California INPATH (//A[@B = "home address"])
```

Finds the document:

```
<A B="home address">San Francisco, California, USA</A>
```

But does not find:

```
<A B="work address">San Francisco, California, USA</A>
```

Path Testing

You can test if a path exists with the `HASPATH` operator. For example, the query:

```
HASPATH(A/B/C)
```

finds and returns a score of 100 for the document

```
<A><B><C>dog</C></B></A>
```

without the query having to reference *dog* at all.

Limitations

Testing for Equality

The following is an example of an `INPATH` equality test.

```
dog INPATH (A[@B = "foo"])
```

The following limitations apply for these expressions:

- Only equality and inequality are supported. Range operators and functions are not supported.
- The left hand side of the equality must be an attribute. Tags and literals here are not enabled.
- The right hand side of the equality must be a literal. Tags and attributes here are not allowed.
- The test for equality depends on your lexer settings. With the default settings, the query

  ```
  dog INPATH (A[@B= "pot of gold"])
  ```

  matches the following sections:

  ```
  <A B="POT OF GOLD">dog</A>
  ```

  and

  ```
  <A B="pot of gold">dog</A>
  ```

  because lexer is case-insensitive by default.

  ```
  <A B="POT IS GOLD">dog</A>
  ```

because *of* and *is* are default stopwords in English, and a stopword matches any stopword word.

```
<A B="POT_OF_GOLD">dog</A>
```

because the underscore character is not a join character by default.

MDATA

Use the MDATA operator to query documents that contain MDATA sections. MDATA sections are metadata that have been added to documents to speed up mixed querying.

MDATA queries are treated exactly as literals. For example, with the query:

```
MDATA(price, $1.24)
```

the *$* is not interpreted as a stem operator, nor is the . (period) transformed into whitespace. A right (close) parenthesis terminates the MDATA operator, so that MDATA values that have close parentheses cannot be searched.

Syntax

Syntax
MDATA(*sectionname*, *value*)

sectionname
The name of the MDATA section(s) to search. MDATA will also search DATE or numerical equality if the sectionname parameter is mapped to a FILTER BY column of DATE or some numerical type.

value
The value of the MDATA section. For example, if an MDATA section called Booktype has been created, it might have a value of *paperback*.

For MDATA operator on MDATA sections that are mapped to a DATE FILTER BY column, the MDATA value must follow the Date format: YYYY-MM-DD HH24:MI:SS. Otherwise, the expected rows will not be returned. If the time component is omitted, it will default to 00:00:00, according to SQL semantics.

Example

Suppose you want to query for books written by the writer *Nigella Lawson* that contain the word *summer*. Assuming that an MDATA section called AUTHOR has been declared, you can query as follows:

```
SELECT id FROM idx_docs
  WHERE CONTAINS(text, 'summer AND MDATA(author, Nigella Lawson)')>0
```

This query will only be successful if an AUTHOR tag has the exact value *Nigella Lawson* (after simplified tokenization). *Nigella* or *Ms. Nigella Lawson* will not work.

Notes

MDATA query values ignore stopwords.

The MDATA operator returns 100 or 0, depending on whether the document is a match.

The MDATA operator is not supported for CTXCAT, CTXRULE, or CTXXPATH indexes.

Table 3–2 shows how MDATA interacts with some other query operators:

Table 3–2 MDATA and Other Query Operators

Operator	Example	Allowed?
AND	dog & MDATA(a, b)	yes
OR	dog | MDATA(a, b)	yes
NOT	dog ~ MDATA(a, b)	yes
MINUS	dog - MDATA(a, b)	yes
ACCUM	dog , MDATA(a, b)	yes
PHRASE	MDATA(a, b) dog	no
NEAR	MDATA(a, b) ; dog	no
WITHIN, HASPATH, INPATH	MDATA(a, b) WITHIN c	no
Thesaurus	MDATA(a, SYN(b))	no
expansion	MDATA(a, $b) MDATA(a, b%) MDATA(a, !b) MDATA(a, ?b)	no (syntactically allowed, but the inner operator is treated as literal text)
ABOUT	ABOUT(MDATA(a,b)) MDATA(ABOUT(a))	no (syntactically allowed, but the inner operator is treated as literal text)

When `MDATA` sections repeat, each instance is a separate and independent value. For instance, the document

```
<AUTHOR>Terry Pratchett</AUTHOR><AUTHOR>Douglas Adams</AUTHOR>
```

can be found with any of the following queries:

```
MDATA(author, Terry Pratchett)
MDATA(author, Douglas Adams)
MDATA(author, Terry Pratchett) and MDATA(author, Douglas Adams)
```

but not any of the following:

```
MDATA(author, Terry Pratchett Douglas Adams)
MDATA(author, Terry Pratchett & Douglas Adams)
MDATA(author, Pratchett Douglas)
```

Related Topics

See also "ADD_MDATA" on page 7-9 and "ADD_MDATA_SECTION" on page 7-12, as well as the Section Searching chapter of the *Oracle Text Application Developer's Guide*.

MINUS (-)

Use the `MINUS` operator to lower the score of documents that contain unwanted noise terms. `MINUS` is useful when you want to search for documents that contain one query term but want the presence of a second term to cause a document to be ranked lower.

Syntax

Syntax	Description
term1-*term2* *term1* minus *term2*	Returns documents that contain *term1*. Calculates score by subtracting the score of *term2* from the score of *term1*. Only documents with positive score are returned.

Example

Suppose a query on the term *cars* always returned high scoring documents about *Ford cars*. You can lower the scoring of the Ford documents by using the expression:

```
'cars - Ford'
```

In essence, this expression returns documents that contain the term *cars* and possibly *Ford*. However, the score for a returned document is the score of *cars* minus the score of *Ford*.

Related Topics

See Also: "NOT (~)" on page 3-40

MNOT

The Mild Not (MNOT) operator is similar to the NOT and MINUS operators. The Mild Not operator returns hits where the the left child is not contained by the right child. Both children can only be TERM or PHRASE nodes.

The semantics can be illustrated with a query of "term1 mnot term1 term2", where the hits for "term1 term2" will be filtered out. For example:

- A document with only *term1* will be returned, with score unchanged.
- A document with only *term1 term2* will not be returned.
- A document with *term1 term1 term2* will be returned, but the score will be calculated using just the first *term1* hit.

The behavior described in the third bullet is different from the behavior of NOT, which does not return this type of document.

The MNOT operator is more specific than the MINUS operator, in that the left child must be contained by the right child. If it is not, the Mild Not operator ignores the right child. Also, for Mild Not, the right child is a true filter, that is, it does not simply subtract the scores of left child and right child.

The MNOT operator has precedence lower than NOT and higher than WITHIN.

Syntax

Syntax	Description
term1 mnot *term1 term2*	Returns docs that contain *term1* unless it is part of the phrase *term1 term2*.
term1 mnot *term2*	Returns all documents that contain *term1*. It will be the same query as just *term1*.

Example

The children of the MNOT operator must be a TERM or PHRASE.

```
SELECT * FROM docs
WHERE CONTAINS(txt, 'term1 mnot term1 term2') >0
```

Related Topics

See Also: "NOT (~)" on page 3-40

Narrower Term (NT, NTG, NTP, NTI)

Use the narrower term operators (NT, NTG, NTP, NTI) to expand a query to include all the terms that have been defined in a thesaurus as the narrower or lower level terms for a specified term. They can also expand the query to include all of the narrower terms for each narrower term, and so on down through the thesaurus hierarchy.

Syntax

Syntax	Description
NT(*term*[(*qualifier*)][,*level*][,*thes*])	Expands a query to include all the lower level terms defined in the thesaurus as narrower terms for term.
NTG(*term*[(*qualifier*)][,*level*][,*thes*])	Expands a query to include all the lower level terms defined in the thesaurus as narrower generic terms for term.
NTP(*term*[(*qualifier*)][,*level*][,*thes*])	Expands a query to include all the lower level terms defined in the thesaurus as narrower partitive terms for term.
NTI(*term*[(*qualifier*)][,*level*][,*thes*])	Expands a query to include all the lower level terms defined in the thesaurus as narrower instance terms for term.

term
Specify the operand for the narrower term operator. term is expanded to include the narrower term entries defined for the term in the thesaurus specified by thes. The number of narrower terms included in the expansion is determined by the value for level. You cannot specify expansion operators in the term argument.

qualifier
Specify a qualifier for term, if term is a homograph (word or phrase with multiple meanings, but the same spelling) that appears in two or more nodes in the same hierarchy branch of thes.

If a qualifier is not specified for a homograph in a narrower term query, the query expands to include all of the narrower terms of all homographic terms.

level
Specify the number of levels traversed in the thesaurus hierarchy to return the narrower terms for the specified term. For example, a level of 1 in an NT query returns all the narrower term entries, if any exist, for the specified term. A level of 2 returns all the narrower term entries for the specified term, as well as all the narrower term entries, if any exist, for each narrower term.

The level argument is optional and has a default value of one (1). Zero or negative values for the level argument return only the original query term.

thes
Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. A thesaurus named DEFAULT *must* exist in the thesaurus tables if you use this default value.

Note: If you specify thes, then you must also specify level.

Examples

The following query returns all documents that contain either the term *cat* or any of the NT terms defined for *cat* in the DEFAULT thesaurus:

```
'NT(cat)'
```

If you specify a thesaurus name, then you must also specify level as in:

```
'NT(cat, 2, mythes)'
```

The following query returns all documents that contain either *fairy tale* or any of the narrower instance terms for *fairy tale* as defined in the DEFAULT thesaurus:

```
'NTI(fairy tale)'
```

That is, if the terms *cinderella* and *snow white* are defined as narrower term instances for *fairy tale*, Oracle Text returns documents that contain *fairy tale, cinderella,* or *snow white.*

Notes

Each hierarchy in a thesaurus represents a distinct, separate branch, corresponding to the four narrower term operators. In a narrower term query, Oracle Text only expands the query using the branch corresponding to the specified narrower term operator.

Related Topics

Browse a thesaurus using procedures in the CTX_THES package.

> **See Also:** CTX_THES.NT in Chapter 12, "CTX_THES Package" for more information on browsing the narrower terms in your thesaurus

NDATA

Use the NDATA operator to find matches that are spelled in a similar way or where rearranging the terms of the specified phrase is useful. It is helpful for finding more accurate results when there are frequent misspellings (or inaccurate orderings) of name data in the document set. This operator can be used only on defined NDATA sections. The NDATA syntax enables you to rank the result set so that documents that contain words with high orthographic similarity are scored higher than documents with lower similarity.

Normalization

A lexer does not process NDATA query phrases. Users can, however, set base letter and alternate spelling attributes for a particular section group containing NDATA sections. Query case is normalized and non-character data (except for white space) is removed (for example, numerical or punctuation).

Syntax

```
ndata(sectionname, phrase [,order][,proximity])
```

Parameter Name	Default Value	Parameter Description
sectionname		Specify the name of a defined NDATA sections to query (that is, section_name)
phrase		Specify the phrase for the name data query. The phrase parameter can be a single word or a phrase, or a string of words in free text format. The score returned is a relevant score. Oracle Text ignores any query operators that are included in phrase. The phrase should be a minimum of two characters in length and should not exceed 4000 characters in length.
order	NOORDER	Specify whether individual tokens (terms) in a query should be matched in-order or in any order. The order parameter provides a primary filter for matching candidate documents. ORDER or O - The query terms are matched in-order. NOORDER o N [DEFAULT] - The query terms are matched in any order.
proximity	NOPROXIMITY	Specify whether the proximity of terms should influence the similarity score of candidate matches. That is, if the proximity parameter is enabled, non-matching additional terms between matching terms will reduce the similarity score of candidate matches. PROXIMITY or P - The similarity score influenced by the proximity of query terms in candidate matches. NOPROXIMITY or N [DEFAULT] - The similarity score is not influenced by the proximity of query terms in candidate matches.

Examples

An NDATA query on an indexed surname section name that matches terms in the query phrase in any order without influencing the similarity score by the proximity of the black and smith terms has the form:

```
SELECT entryid, SCORE(1) FROM people WHERE
CONTAINS(idx_column, 'NDATA(surname, black smith)',1)>0;
```

An NDATA query on an indexed surname section name that matches terms in the query phrase in any order and in which similarity scores are influenced by the proximity of the black and smith terms has the form:

```
SELECT entryid, SCORE(1) FROM people WHERE
CONTAINS(idx_column, 'NDATA(surname, black smith,,proximity)',1)>0;
```

An NDATA query on an indexed surname section name that matches terms in the query phrase in-order without influencing the similarity score by the proximity of the black and smith terms has the form:

```
SELECT entryid, SCORE(1) FROM people WHERE
CONTAINS(idx_column, 'NDATA(surname, black smith, order)',1)>0;
```

An NDATA query on an indexed surname section name that matches terms in the query phrase in-order and in which similarity scores are influenced by the proximity of the black and smith terms has the form:

```
SELECT entryid, SCORE(1) FROM people WHERE
CONTAINS(idx_column, 'NDATA(surname, black smith, order, proximity)',1)>0;
```

Notes

The NDATA query operator does not provide offset information. As such, it cannot be used as a child of WITHIN, NEAR(;), or EQUIV(=), and NDATA sections will be ignored by CTX_DOC.HIGHLIGHT, CTX_DOC.SNIPPET, and CTX_DOC.MARKUP. The NDATA operator also is not supported in the CTXCAT grammar. It can be used with other operators, including OR and query templates.

A use case of the NDATA operator may involve finding a particular entry based on an approximate spelling of a person's full-name and an estimated date-of-birth. Supposing the entries' date-of-births are stored as an SDATA section, user-defined scoring's alternate scoring template can be used to combine the scores of the full-name's NDATA section data and the date-of-birth's SDATA section data.

The name john smith is queried for the section specified by the fullname section_name. Altering the NDATA operator's score based on the closeness of the SDATA section's date-of-birth to the date 08-NOV-2005 modifies the ranking of matching documents:

```
<query>
   <textquery grammar="CONTEXT" lang="english">
       NDATA(fullname, john smith)
   </textquery>
   <score algorithm="COUNT" normalization_expr =
       "doc_score-(DATE(8-NOV-2005)-sdata:dob)"/>
</query>
```

NEAR (;)

Use the NEAR operator to return a score based on the proximity of two or more query terms. Oracle Text returns higher scores for terms closer together and lower scores for terms farther apart in a document.

Note: The NEAR operator works with only word queries. You cannot use NEAR in ABOUT queries.

Syntax

Syntax

NEAR((*word1*, *word2*,..., *wordn*) [, max_span [, order]])

Backward compatibility syntax: *word1; word2*

word *1-n*
Specify the terms in the query separated by commas. The query terms can be single words or phrases and may make use of other query operators (see "NEAR with Other Operators").

max_span
Optionally specify the size of the biggest clump. The default is 100. Oracle Text returns an error if you specify a number greater than 100.

A clump is the smallest group of words in which all query terms occur. All clumps begin and end with a query term.

For near queries with two terms, max_span is the maximum distance allowed between the two terms. For example, to query on *dog* and *cat* where *dog* is within 6 words of *cat*, enter the following query:

```
'near((dog, cat), 6)'
```

order
Specify TRUE for Oracle Text to search for terms in the order you specify. The default is FALSE.

For example, to search for the words *monday*, *tuesday*, and *wednesday* in that order with a maximum clump size of 20, enter the following query:

```
'near((monday, tuesday, wednesday), 20, TRUE)'
```

Note: To specify order, then you must always specify a number for max_span.

Oracle Text might return different scores for the same document when you use identical query expressions that have the order flag set differently. For example, Oracle Text might return different scores for the same document when you enter the following queries:

```
'near((dog, cat), 50, FALSE)'
'near((dog, cat), 50, TRUE)'
```

NEAR Scoring

The scoring for the NEAR operator combines frequency of the terms with proximity of terms. For each document that satisfies the query, Oracle Text returns a score between 1 and 100 that is proportional to the number of clumps in the document and inversely proportional to the average size of the clumps. This means many small clumps in a document result in higher scores, because small clumps imply closeness of terms.

The number of terms in a query also affects score. Queries with many terms, such as seven, generally need fewer clumps in a document to score 100 than do queries with few terms, such as two.

A *clump* is the smallest group of words in which all query terms occur. All clumps begin and end with a query term. Define clump size with the max_span parameter, as described in this section.

The size of a clump does not include the query terms themselves. So for the query NEAR((DOG, CAT), 1), *dog cat* will be a match, and *dog ate cat* will be a match, but *dog sat on cat* will *not* be a match.

NEAR with Other Operators

You can use the NEAR operator with other operators such as AND and OR. Scores are calculated in the regular way.

For example, to find all documents that contain the terms *tiger*, *lion*, and *cheetah* where the terms *lion* and *tiger* are within 10 words of each other, enter the following query:

```
'near((lion, tiger), 10) AND cheetah'
```

The score returned for each document is the lower score of the near operator and the term *cheetah*.

You can also use the equivalence operator to substitute a single term in a near query:

```
'near((stock crash, Japan=Korea), 20)'
```

This query asks for all documents that contain the phrase *stock crash* within twenty words of *Japan* or *Korea*.

The following NEAR syntax is now valid:

```
SELECT * FROM docs WHERE CONTAINS(txt, 'near((aterm1 aterm2 ... atermI
OR bterm1 bterm2 ... btermJ
OR cterm1 cterm2 ... ctermK, dterm))') >0
```

There can be any number of ORs in a given NEAR child, and the OR can appear in any of the NEAR children.

The NEAR within NEAR feature allows users to use nested proximity queries. Users can execute queries such as the following:

```
SELECT * FROM docs
WHERE CONTAINS(txt, 'near((near((term1, term2),5), term3), 100)')>0
```

This will return documents where *term1*, *term2*, and *term3* are near within a 100 token window and, additionally, the tokens *term1* and *term2* are near within a 5 token window.

Mixing the semicolon and NEAR syntax is not supported and will throw an error. That is, the queries "near((a;b,c),3)" or "near((a,b));c" will be disallowed.

The following operators also work with NEAR and ; :

- EQUIV
- All expansion operators that produce words, phrases, or EQUIV. These include:
  - soundex
  - fuzzy
  - wildcards
  - stem

Backward Compatibility NEAR Syntax

You can write near queries using the syntax of previous Oracle Text releases. For example, to find all documents where *lion* occurs near *tiger*, write:

```
'lion near tiger'
```

or with the semi-colon as follows:

```
'lion;tiger'
```

This query is equivalent to the following query:

```
'near((lion, tiger), 100, FALSE)'
```

Note: Only the syntax of the NEAR operator is backward compatible. In the example, the score returned is calculated using the clump method as described in this section.

Highlighting with the NEAR Operator

When you use highlighting and your query contains the near operator, all occurrences of all terms in the query that satisfy the proximity requirements are highlighted. Highlighted terms can be single words or phrases.

For example, assume a document contains the following text:

```
Chocolate and vanilla are my favorite ice cream flavors.  I like chocolate served
in a waffle cone, and vanilla served in a cup with carmel syrup.
```

If the query is *near((chocolate, vanilla)), 100, FALSE)*, the following is highlighted:

```
<<Chocolate>> and <<vanilla>> are my favorite ice cream flavors.  I like
<<chocolate>> served in a waffle cone, and <<vanilla>> served in a cup with
caramel syrup.
```

However, if the query is *near((chocolate, vanilla)), 4, FALSE)*, only the following is highlighted:

```
<<Chocolate>> and <<vanilla>> are my favorite ice cream flavors.  I like
chocolate served in a waffle cone, and vanilla served in a cup with carmel syrup.
```

See Also: Chapter 8, "CTX_DOC Package" for more information about the procedures for highlighting

Section Searching and NEAR

Use the NEAR operator with the WITHIN operator for section searching as follows:

```
'near((dog, cat), 10) WITHIN Headings'
```

When evaluating expressions such as these, Oracle Text looks for clumps that lie entirely within the given section.

In this example, only those clumps that contain *dog* and *cat* that lie entirely within the section *Headings* are counted. That is, if the term *dog* lies within *Headings* and the term *cat* lies five words from *dog*, but outside of *Headings*, this pair of words does not satisfy the expression and is not counted.

NOT (~)

Use the NOT operator to search for documents that contain one query term and not another.

Syntax

Syntax	Description
term1~*term2*	Returns documents that contain *term1* and not *term2*.
term1 not *term2*	

Examples

To obtain the documents that contain the term *animals* but not *dogs*, use the following expression:

```
'animals ~ dogs'
```

Similarly, to obtain the documents that contain the term *transportation* but not *automobiles* or *trains*, use the following expression:

```
'transportation not (automobiles or trains)'
```

Note: The NOT operator does not affect the scoring produced by the other logical operators.

Related Topics

See Also: "MINUS (-)" on page 3-30

OR (|)

Use the OR operator to search for documents that contain at least one occurrence of *any* of the query terms.

Syntax

Syntax	Description
term1 | *term2* *term1* or *term2*	Returns documents that contain *term1* or *term2*. Returns the maximum score of its operands. At least one term must exist; higher score taken.

Examples

To obtain the documents that contain the term *cats* or the term *dogs*, use either of the following expressions:

```
'cats | dogs'
'cats OR dogs'
```

Scoring

In an OR query, the score returned is the score for the highest query term. In the example, if the scores for *cats* and *dogs* is 30 and 40 within a document, the document scores 40.

Related Topics

See Also: The OR operator returns documents that contain *any* of the query terms, while the AND operator returns documents that contain *all* query terms. See "AND (&)" on page 3-9.

Preferred Term (PT)

Use the preferred term operator (PT) to replace a term in a query with the preferred term that has been defined in a thesaurus for the term.

Syntax

Syntax	Description
PT(*term*[,*thes*])	Replaces the specified word in a query with the preferred term for *term*.

term
Specify the operand for the preferred term operator. term is replaced by the preferred term defined for the term in the specified thesaurus. However, if no PT entries are defined for the term, term is not replaced in the query expression and term is the result of the expansion.

You cannot specify expansion operators in the term argument.

thes
Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. As a result, a thesaurus named DEFAULT *must* exist in the thesaurus tables before using any of the thesaurus operators.

Example

The term *automobile* has a preferred term of *car* in a thesaurus. A PT query for *automobile* returns all documents that contain the word *car*. Documents that contain the word *automobile* are not returned.

Related Topics

Browse a thesaurus using procedures in the CTX_THES package.

> **See Also:** CTX_THES.PT in Chapter 12, "CTX_THES Package" form more information on browsing the preferred terms in your thesaurus

Related Term (RT)

Use the related term operator (RT) to expand a query to include all related terms that have been defined in a thesaurus for the term.

Syntax

Syntax	Description
RT(*term*[,*thes*])	Expands a query to include all the terms defined in the thesaurus as a related term for term.

term
Specify the operand for the related term operator. term is expanded to include term and all the related entries defined for term in thes.

You cannot specify expansion operators in the term argument.

thes
Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. As a result, a thesaurus named DEFAULT *must* exist in the thesaurus tables before using any of the thesaurus operators.

Example

The term *dog* has a related term of *wolf*. An RT query for *dog* returns all documents that contain the word *dog* and *wolf*.

Related Topics

Browse a thesaurus using procedures in the CTX_THES package

> **See Also:** CTX_THES.RT in Chapter 12, "CTX_THES Package" for more information on browsing the related terms in your thesaurus

SDATA

Use the SDATA operator to perform tests on SDATA sections and columns, which contain structured data values. SDATA sections speed up mixed querying and ordering. This operator provides structured predicate support for CONTAINS, which extends non-SQL interfaces such as count_hits or the result set interface.

SDATA operators should only be used as descendants of AND operators that also have non-SDATA children.

SDATA queries perform on string or numeric literals, and on date strings. The string literal and date string are enclosed within single or double quote characters. The numeric value is not enclosed in quote characters, and must conform to the SQL format of NUMBER. For example:

```
CONTAINS(text, "dog and SDATA(category = ''news'')")>0 ...

SDATA(rating between 1.2 and 3.4) ...

SDATA(author LIKE 'FFORDE%') ...

SDATA(date >='2005-09-18') ...
```

Closed parentheses are permitted, as long as they are enclosed in single or double quotes.

The SDATA operator can be used in query templates.

Syntax

Syntax	
SData	:= "SDATA" "(" SDataPredicate ")"
SDataPredicate	:= *sectionname* SDataTest
SDataTest	:= <SDataSingleOp SDataLiteral> | SDataBetweenOp | <"is" ("not")? "null">
SDataSingleOp	:= ("<" | "<=" | "=" | ">=" | ">" | "!=" | "<>" | "like") SDataLiteral
SDataBetweenOp	:= "between" SDataLiteral "and" SDataLiteral
SDataLiteral	:= *numeric_literal* | """ *string_literal* """ | """ *date_string* """

sectionname
The name of the SDATA section(s) on which to search and perform the test, or check.

SDataLiteral
The value of the SDATA section. This must be either a string literal, numeric literal, or a date string.

The SDATA operator returns a score of 100 if the enclosed predicate returns TRUE, and returns 0 otherwise. In the case of a NULL value, the SDATA operator returns a score of 0 (since in SQL it would not return TRUE).

Multi-valued semantics are not defined, as multi-valued SDATA sections are not supported.

Comparison of strings is case sensitive. The BINARY collation is always used.

Note: For the SDATA operator on SDATA sections that are mapped to a DATE FILTER BY column, the SDATA value must follow the Date format: YYYY-MM-DD or YYYY-MM-DD HH24:MI:SS. Otherwise, the expected rows will not be returned. If the time component is omitted, it will default to 00:00:00, according to SQL semantics. This Date format is always used, regardless of the setting of the NLS_DATE_FORMAT environment variable.

Examples

Suppose that you want to query for books in the fiction category that contain the word *summer*. Assuming that an SDATA section called CATEGORY has been declared, you can query as follows:

```
SELECT id FROM idx_docs
  WHERE CONTAINS(text, 'summer AND SDATA(category = "fiction")')>0
```

Restrictions

- An error is raised if the section name is not a defined SDATA section. The source of the section (for example, tag versus column) is not important.
- The syntax precludes RHS SDATA and expressions.
- SDATA operators cannot be children of WITHIN, INPATH, HASPATH, or NEAR.
- The datatype of the named SDATA section must be compatible with the literal provided (and the operator, for example, LIKE) or an error is raised.
- SDATA operators are not supported in CTXRULE query documents.
- SDATA operators have no effect on highlighting.

Notes

Oracle recommends using SDATA operators only as descendants of AND operators that also have non-SDATA children. Essentially, use SDATA operators as secondary (that is, checking or non-driving) criteria. For instance, "find documents with DOG that also have price > 5", rather than "find documents with rating > 4". Other usage may operate properly, but may not have optimal performance.

The following examples are consistent with recommended use:

```
dog & SDATA(foo = 5)
```

The SDATA is a child of an AND operator that also has non-SDATA children.

```
dog & (SDATA(foo = 5) | SDATA(x = 1))
```

Although the SDATA operators here are children of OR, they are still descendants of an AND operator with non-SDATA children.

The following examples show use that is not recommended:

```
SDATA(foo = 5)
```

Here, SDATA is the only criteria and, therefore, the driving criteria.

```
dog | SDATA(bar = 9)
```

The SDATA in this example is a child of an OR operator rather than an AND.

```
SDATA(foo = 5) & SDATA(bar = 7)
```

While both SDATA operators in this example are descendants of AND, this AND operator does not have non-SDATA children.

Related Topics

ADD_SDATA_COLUMN on page 7-14

ADD_SDATA_SECTION on page 7-16

CTX_SECTIONS on page G-9 in Appendix G, "Oracle Text Views"

See Also:

- *Oracle Database SQL Language Reference*
- Chapter 8, "Searching Document Sections in Oracle Text" in *Oracle Text Application Developer's Guide*

soundex (!)

Use the soundex (!) operator to expand queries to include words that have similar sounds; that is, words that sound like other words. This function enables comparison of words that are spelled differently, but sound alike in English.

Syntax

Syntax	Description
!*term*	Expands a query to include all terms that sound the same as the specified term (English-language text only).

Example

```
SELECT ID, COMMENT FROM EMP_RESUME
WHERE CONTAINS (COMMENT, '!SMYTHE') > 0 ;

ID COMMENT
-- ------------
23 Smith is a hard worker who..
```

Language

Soundex works best for languages that use a 7-bit character set, such as English. It can be used, with lesser effectiveness, for languages that use an 8-bit character set, such as many Western European languages.

If you have base-letter conversion specified for a text column and the query expression contains a soundex operator, then Oracle Text operates on the base-letter form of the query.

stem ($)

Use the stem ($) operator to search for terms that have the same linguistic root as the query term.

If you use the BASIC_LEXER to index your language, stemming performance can be improved by using the index_stems attribute.

The Oracle Text stemmer, licensed from XSoft Division of Xerox Corporation, supports the following languages with the BASIC_LEXER: English, French, Spanish, Italian, German, and Dutch.

Japanese stemming is supported with the JAPANESE_LEXER.

Specify your stemming language with the BASIC_WORDLIST wordlist preference.

Syntax

Syntax	Description
$*term*	Expands a query to include all terms having the same stem or root word as the specified term.

Examples

Input	Expands To
$scream	scream screaming screamed
$distinguish	distinguish distinguished distinguishes
$guitars	guitars guitar
$commit	commit committed
$cat	cat cats
$sing	sang sung sing

Behavior with Stopwords

If stem returns a word designated as a stopword, the stopword is not included in the query or highlighted by CTX_QUERY.HIGHLIGHT or CTX_QUERY.MARKUP.

Related Topics

See Also: For more information about enabling the stem operator with BASIC_LEXER, see "BASIC_LEXER" in Chapter 2, "Oracle Text Indexing Elements".

Stored Query Expression (SQE)

Use the SQE operator to call a stored query expression created with the `CTX_QUERY.STORE_SQE` procedure.

Stored query expressions can be used for creating predefined bins for organizing and categorizing documents or to perform iterative queries, in which an initial query is refined using one or more additional queries.

Syntax

Syntax	Description
SQE(*SQE_name*)	Returns the results for the stored query expression *SQE_name*.

Examples

To create an SQE named *disasters*, use `CTX_QUERY.STORE_SQE` as follows:

```
begin
ctx_query.store_sqe('disasters', 'hurricane or earthquake or blizzard');
end;
```

This stored query expression returns all documents that contain either *hurricane*, *earthquake* or *blizzard*.

This SQE can then be called within a query expression as follows:

```
SELECT SCORE(1), docid FROM news
WHERE CONTAINS(resume, 'sqe(disasters)', 1)> 0
ORDER BY SCORE(1);
```

Limitations

Up to 100 stored query expressions (SQEs) can be stored in a single Text query. If a Text query has more than 100 SQEs, including nested SQEs, then the query fails and error DRG-50949 is raised.

SYNonym (SYN)

Use the synonym operator (SYN) to expand a query to include all the terms that have been defined in a thesaurus as synonyms for the specified term.

Syntax

Syntax	Description
SYN(*term*[,*thes*])	Expands a query to include all the terms defined in the thesaurus as synonyms for term.

term
Specify the operand for the synonym operator. term is expanded to include term and all the synonyms defined for term in thes.

You cannot specify expansion operators in the term argument.

thes
Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. A thesaurus named DEFAULT must exist in the thesaurus tables if you use this default value.

Examples

The following query expression returns all documents that contain the term *dog* or any of the synonyms defined for *dog* in the DEFAULT thesaurus:

```
'SYN(dog)'
```

Compound Phrases in Synonym Operator

Expansion of compound phrases for a term in a synonym query are returned as AND conjunctives.

For example, the compound phrase *temperature* + *measurement* + *instruments* is defined in a thesaurus as a synonym for the term *thermometer*. In a synonym query for *thermometer*, the query is expanded to:

```
{thermometer} OR ({temperature}&{measurement}&{instruments})
```

Related Topics

Browse your thesaurus using procedures in the CTX_THES package.

> **See Also:** CTX_THES.SYN in Chapter 12, "CTX_THES Package" for more information on browsing the synonym terms in your thesaurus

threshold (>)

Use the threshold operator (>) in two ways:

- at the expression level
- at the query term level

The threshold operator at the expression level eliminates documents in the result set that score below a threshold number.

The threshold operator at the query term level selects a document based on how a term scores in the document.

Syntax

Syntax	Description
expression>n	Returns only those documents in the result set that score above the threshold *n*.
term>n	Within an expression, returns documents that contain the query term with score of at least *n*.

Examples

At the expression level, to search for documents that contain *relational databases* and to return only documents that score greater than 75, use the following expression:

```
'relational databases > 75'
```

At the query term level, to select documents that have at least a score of 30 for *lion* and contain *tiger,* use the following expression:

```
'(lion > 30) and tiger'
```

Translation Term (TR)

Use the translation term operator (TR) to expand a query to include all defined foreign language equivalent terms.

Syntax

Syntax	Description
TR(*term*[, *lang*, [*thes*]])	Expands *term* to include all the foreign equivalents that are defined for *term*.

term
Specify the operand for the translation term operator. term is expanded to include all the foreign language entries defined for term in thes. You cannot specify expansion operators in the term argument.

lang
Optionally, specify which foreign language equivalents to return in the expansion. The language you specify must match the language as defined in thes. (You may specify only one language at a time.) If you omit this parameter or specify it as ALL, the system expands to use all defined foreign language terms.

thes
Optionally, specify the name of the thesaurus used to return the expansions for the specified term. The thes argument has a default value of DEFAULT. As a result, a thesaurus named DEFAULT *must* exist in the thesaurus tables before you can use any of the thesaurus operators.

> **Note:** If you specify thes, then you must also specify lang.

Examples

Consider a thesaurus MY_THES with the following entries for *cat*:

```
cat
  SPANISH: gato
  FRENCH:  chat
```

To search for all documents that contain *cat* and the spanish translation of *cat*, enter the following query:

```
'tr(cat, spanish, my_thes)'
```

This query expands to:

```
'{cat}|{gato}'
```

Related Topics

Browse a thesaurus using procedures in the CTX_THES package.

> **See Also:** CTX_THES.TR in Chapter 12, "CTX_THES Package" for more information on browsing the related terms in your thesaurus

Translation Term Synonym (TRSYN)

Use the translation term operator (TR) to expand a query to include all the defined foreign equivalents of the query term, the synonyms of query term, and the foreign equivalents of the synonyms.

Syntax

Syntax	Description
TRSYN(*term*[, *lang*, [*thes*]])	Expands term to include foreign equivalents of term, the synonyms of term, and the foreign equivalents of the synonyms.

term
Specify the operand for this operator. term is expanded to include all the foreign language entries and synonyms defined for term in thes. You cannot specify expansion operators in the term argument.

lang
Optionally, specify which foreign language equivalents to return in the expansion. The language you specify must match the language as defined in thes. If you omit this parameter, the system expands to use all defined foreign language terms.

thes
Optionally, specify the name of the thesaurus used to return the expansions for the specified term. The thes argument has a default value of DEFAULT. As a result, a thesaurus named DEFAULT *must* exist in the thesaurus tables before you can use any of the thesaurus operators.

> **Note:** If you specify thes, then you must also specify lang.

Examples

Consider a thesaurus MY_THES with the following entries for *cat*:

```
cat
  SPANISH: gato
  FRENCH:  chat
  SYN lion
    SPANISH: leon
```

To search for all documents that contain *cat*, the spanish equivalent of *cat*, the synonym of *cat*, and the spanish equivalent of *lion*, enter the following query:

```
'trsyn(cat, spanish, my_thes)'
```

This query expands to:

```
'{cat}|{gato}|{lion}|{leon}'
```

Related Topics

Browse a thesaurus using procedures in the CTX_THES package.

See Also: CTX_THES.TRSYN in Chapter 12, "CTX_THES Package" for more information on browsing the translation and synonym terms in your thesaurus

Top Term (TT)

Use the top term operator (TT) to replace a term in a query with the *top term* that has been defined for the term in the standard hierarchy (Broader Term [BT], Narrower Term [NT]) in a thesaurus. A top term is the broadest conceptual term related to a given query term. For example, a thesaurus might define the following hierarchy:

```
DOG
   BT1 CANINE
     BT2 MAMMAL
       BT3 VERTEBRATE
         BT4 ANIMAL
```

The top term for *dog* in this thesaurus is *animal*.

Top terms in the generic (BTG, NTG), partitive (BTP, NTP), and instance (BTI, NTI) hierarchies are not returned.

Syntax

Syntax	Description
TT(*term*[,*thes*])	Replaces the specified word in a query with the top term in the standard hierarchy (BT, NT) for term.

term
Specify the operand for the top term operator. term is replaced by the top term defined for the term in the specified thesaurus. However, if no TT entries are defined for term, term is not replaced in the query expression and term is the result of the expansion.

You cannot specify expansion operators in the term argument.

thes
Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. A thesaurus named DEFAULT must exist in the thesaurus tables if you use this default value.

Example

The term *dog* has a top term of *animal* in the standard hierarchy of a thesaurus. A TT query for *dog* returns all documents that contain the phrase *animal*. Documents that contain the word *dog* are not returned.

Related Topics

Browse your thesaurus using procedures in the CTX_THES package.

> **See Also:** CTX_THES.TT on page 12-46 for more information on browsing the top terms in your thesaurus

weight (*)

The weight operator multiplies the score by the given factor, topping out at 100 when the score exceeds 100. For example, the query *cat, dog*2* sums the score of *cat* with twice the score of *dog*, topping out at 100 when the score is greater than 100.

In expressions that contain more than one query term, use the weight operator to adjust the relative scoring of the query terms. Reduce the score of a query term by using the weight operator with a number less than 1; increase the score of a query term by using the weight operator with a number greater than 1 and less than 10.

The weight operator is useful in ACCUMulate (,), AND (&), or OR (|) queries when the expression has more than one query term. With no weighting on individual terms, the score cannot tell which of the query terms occurs the most. With term weighting, you can alter the scores of individual terms and hence make the overall document ranking reflect the terms you are interested in.

Syntax

Syntax	Description
*term*n*	Returns documents that contain `term`. Calculates score by multiplying the raw score of `term` by `n`, where `n` is a number from 0.1 to 10.

Examples

Suppose you have a collection of sports articles. You are interested in the articles about Brazilian soccer. It turns out that a regular query on *soccer or Brazil* returns many high ranking articles on US soccer. To raise the ranking of the articles on Brazilian soccer, enter the following query:

```
'soccer or Brazil*3'
```

Table 3–3 illustrates how the weight operator can change the ranking of three hypothetical documents A, B, and C, which all contain information about soccer. The columns in the table show the total score of four different query expressions on the three documents.

Table 3–3 Score Samples

	soccer	Brazil	soccer or Brazil	soccer or Brazil*3
A	20	10	20	30
B	10	30	30	90
C	50	20	50	60

The score in the third column containing the query *soccer or Brazil* is the score of the highest scoring term. The score in the fourth column containing the query *soccer or Brazil*3* is the larger of the score of the first column *soccer* and of the score *Brazil* multiplied by three, *Brazil*3*.

With the initial query of *soccer or Brazil*, the documents are ranked in the order C B A. With the query of *soccer or Brazil*3*, the documents are ranked B C A, which is the preferred ranking.

Weights can be added to multiple terms. The query *Brazil OR (soccer AND Brazil)*3* will increase the relative scores for documents that contain both *soccer* and *Brazil*.

wildcards (% _)

Wildcard characters can be used in query expressions to expand word searches into pattern searches. The wildcard characters are:

Wildcard Character	Description
%	The percent wildcard can appear any number of times at any part of the search term. The search term will be expanded into an equivalence list of terms. The list consists of all terms in the index that match the wildcarded term, with zero or more characters in place of the percent character.
_	The underscore wildcard specifies a single position in which any character can occur.

The total number of wildcard expansions from all words in a query containing unescaped wildcard characters cannot exceed the maximum number of expansions specified by the `BASIC_WORDLIST` attribute `WILDCARD_MAXTERMS`. For more information, see "BASIC_WORDLIST" on page 3-2.

> **Note:** When a wildcard expression translates to a stopword, the stopword is not included in the query and not highlighted by CTX_DOC.HIGHLIGHT or CTX_DOC.MARKUP.

Right-Truncated Queries

Right truncation involves placing the wildcard on the right-hand-side of the search string.

For example, the following query expression finds all terms beginning with the pattern *scal*:

```
'scal%'
```

Left- and Double-Truncated Queries

Left truncation involves placing the wildcard on the left-hand-side of the search string.

To find words such as *king*, *wing* or *sing*, write the query as follows:

```
'_ing'
```

For all words that end with *ing*, enter:

```
'%ing'
```

Combine left-truncated and right-truncated searches to create double-truncated searches. The following query finds all documents that contain words that contain the substring *%benz%*

```
'%benz%'
```

Improving Wildcard Query Performance

Improve wildcard query performance by adding a substring or prefix index.

When your wildcard queries are left- and double-truncated, you can improve query performance by creating a substring index. Substring indexes improve query

performance for all types of left-truncated wildcard searches such as *%ed*, *_ing*, or *%benz%*.

When your wildcard queries are right-truncated, you can improve performance by creating a prefix index. A prefix index improves query performance for wildcard searches such as *to%*.

> **See Also:** "BASIC_WORDLIST" on page 2-57 in Chapter 2, "Oracle Text Indexing Elements" for more information about creating substring and prefix indexes

WITHIN

Use the WITHIN operator to narrow a query down into document sections. Document sections can be one of the following:

- Zone sections
- Field sections
- Attribute sections
- Special sections (sentence or paragraph)

Syntax

Syntax	Description
expression WITHIN *section*	Searches for expression within the pre-defined zone, field, or attribute section. If section is a zone, expression can contain one or more WITHIN operators (nested WITHIN) whose section is a zone or special section. If section is a field or attribute section, expression cannot contain another WITHIN operator.
expression WITHIN SENTENCE	Searches for documents that contain expression within a sentence. Specify an AND or NOT query for expression. The expression can contain one or more WITHIN operators (nested WITHIN) whose section is a zone or special section.
expression WITHIN PARAGRAPH	Searches for documents that contain expression within a paragraph. Specify an AND or NOT query for expression. The expression can contain one or more WITHIN operators (nested WITHIN) whose section is a zone or special section.

WITHIN Limitations

The WITHIN operator has the following limitations:

- You cannot embed the WITHIN clause in a phrase. For example, you cannot write: *term1 WITHIN section term2*
- Because WITHIN is a reserved word, you must escape the word with braces to search on it.

WITHIN Operator Examples

Querying Within Zone Sections

To find all the documents that contain the term *San Francisco* within the section *Headings*, write the query as follows:

```
'San Francisco WITHIN Headings'
```

To find all the documents that contain the term *sailing* and contain the term *San Francisco* within the section *Headings*, write the query in one of two ways:

```
'(San Francisco WITHIN Headings) and sailing'

'sailing and San Francisco WITHIN Headings'
```

Compound Expressions with WITHIN

To find all documents that contain the terms *dog* and *cat* within the same section *Headings*, write the query as follows:

```
'(dog and cat) WITHIN Headings'
```

This query is logically different from:

```
'dog WITHIN Headings and cat WITHIN Headings'
```

This query finds all documents that contain *dog* and *cat* where the terms *dog* and *cat* are in *Headings* sections, regardless of whether they occur in the same *Headings* section or different sections.

Near with WITHIN

To find all documents in which *dog* is near *cat* within the section *Headings*, write the query as follows:

```
'dog near cat WITHIN Headings'
```

> **Note:** The near operator has higher precedence than the `WITHIN` operator so braces are not necessary in this example. This query is equivalent to *(dog near cat) WITHIN Headings*.

Nested WITHIN Queries

You can nest the within operator to search zone sections within zone sections.

For example, assume that a document set had the zone section `AUTHOR` nested within the zone `BOOK` section. Write a nested `WITHIN` query to find all occurrences of *scott* within the `AUTHOR` section of the `BOOK` section as follows:

```
'(scott WITHIN AUTHOR) WITHIN BOOK'
```

Querying Within Field Sections

The syntax for querying within a field section is the same as querying within a zone section. The syntax for most of the examples given in the previous section, "Querying Within Zone Sections", apply to field sections.

However, field sections behave differently from zone sections in terms of

- Visibility: Make text within a field section invisible.
- Repeatability: `WITHIN` queries cannot distinguish repeated field sections.
- Nestability: You cannot enter a nested `WITHIN` query with a field section.

The following sections describe these differences.

Visible Flag in Field Sections

When a field section is created with the visible flag set to `FALSE` in `CTX_DDL.ADD_FIELD_SECTION`, the text within a field section can only be queried using the `WITHIN` operator.

For example, assume that TITLE is a field section defined with visible flag set to FALSE. Then the query *dog* without the WITHIN operator will *not* find a document containing:

```
<TITLE>The dog</TITLE> I like my pet.
```

To find such a document, use the WITHIN operator as follows:

```
'dog WITHIN TITLE'
```

Alternatively, set the visible flag to TRUE when you define TITLE as a field section with CTX_DDL.ADD_FIELD_SECTION.

> **See Also:** "ADD_FIELD_SECTION" in Chapter 7, "CTX_DDL Package" for more information about creating field sections

Repeated Field Sections

WITHIN queries *cannot* distinguish repeated field sections in a document. For example, consider the document with the repeated section <author>:

```
<author> Charles Dickens </author>
<author> Martin Luther King </author>
```

Assuming that <author> is defined as a field section, a query such as (*charles and martin) within author* returns the document, even though these words occur in separate tags.

To have WITHIN queries distinguish repeated sections, define the sections as zone sections.

Nested Field Sections

You cannot enter a nested WITHIN query with field sections. Doing so raises an error.

Querying Within Sentence or Paragraphs

Querying within sentence or paragraph boundaries is useful to find combinations of words that occur in the same sentence or paragraph. To query sentence or paragraphs, you must first add the special section to your section group before you index. Do so with CTX_DDL.ADD_SPECIAL_SECTION.

To find documents that contain *dog* and *cat* within the same sentence:

```
'(dog and cat) WITHIN SENTENCE'
```

To find documents that contain *dog* and *cat* within the same paragraph:

```
'(dog and cat) WITHIN PARAGRAPH'
```

To find documents that contain sentences with the word *dog* but not *cat*:

```
'(dog not cat) WITHIN SENTENCE'
```

Querying Within Attribute Sections

Query within attribute sections when you index with either XML_SECTION_GROUP or AUTO_SECTION_GROUP as your section group type.

Assume you have an XML document as follows:

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

Define the section title@book to be the attribute section title. Do so with the CTX_DLL.ADD_ATTR_SECTION procedure or dynamically after indexing with ALTER INDEX.

> **Note:** When you use the AUTO_SECTION_GROUP to index XML documents, the system automatically creates attribute sections and names them in the form attribute@tag.
>
> If you use the XML_SECTION_GROUP, you can name attribute sections anything with CTX_DDL.ADD_ATTR_SECTION.

To search on *Tale* within the attribute section title, enter the following query:

```
'Tale WITHIN title'
```

Constraints for Querying Attribute Sections

The following constraints apply to querying within attribute sections:

- Regular queries on attribute text do not hit the document unless qualified in a within clause. Assume you have an XML document as follows:

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

A query on *Tale* by itself does not produce a hit on the document unless qualified with WITHIN title@book. (This behavior is like field sections when you set the visible flag set to false.)

- You cannot use attribute sections in a nested WITHIN query.
- Phrases ignore attribute text. For example, if the original document looked like:

```
Now is the time for all good <word type="noun"> men </word> to come to the aid.
```

Then this document would hit on the regular query *good men*, ignoring the intervening attribute text.

- WITHIN queries can distinguish repeated attribute sections. This behavior is like zone sections but unlike field sections. For example, you have a document as follows:

```
<book title="Tale of Two Cities">It was the best of times.</book>
<book title="Of Human Bondage">The sky broke dull and gray.</book>
```

Assume that book is a zone section and book@author is an attribute section. Consider the query:

```
'(Tale and Bondage) WITHIN book@author'
```

This query does *not* hit the document, because *tale* and *bondage* are in different occurrences of the attribute section book@author.

Notes

Section Names

The WITHIN operator requires you to know the name of the section you search. A list of defined sections can be obtained using the CTX_SECTIONS or CTX_USER_SECTIONS views.

Section Boundaries

For special and zone sections, the terms of the query must be fully enclosed in a particular occurrence of the section for the document to satisfy the query. This is not a requirement for field sections.

For example, consider the query where *bold* is a zone section:

```
'(dog and cat) WITHIN bold'
```

This query finds:

```
<B>dog cat</B>
```

but it does not find:

```
<B>dog</B><B>cat</B>
```

This is because dog and cat must be in the same *bold* section.

This behavior is especially useful for special sections, where

```
'(dog and cat) WITHIN sentence'
```

means find *dog* and *cat* within the same sentence.

Field sections on the other hand are meant for non-repeating, embedded metadata such as a title section. Queries within field sections cannot distinguish between occurrences. All occurrences of a field section are considered to be parts of a single section. For example, the query:

```
(dog and cat) WITHIN title
```

can find a document like this:

<TITLE>dog</TITLE><TITLE>cat</TITLE>

In return for this field section limitation and for the overlap and nesting limitations, field section queries are generally faster than zone section queries, especially if the section occurs in every document, or if the search term is common.

4

Special Characters in Oracle Text Queries

This chapter describes the special characters that can be used in Text queries. In addition, it provides a list of the words and characters that Oracle Text treats as reserved words and characters.

The following topics are covered in this chapter:

Grouping Characters

The grouping characters control operator precedence by grouping query terms and operators in a query expression. The grouping characters are:

Table 4–1 Characters for Grouping Query Terms

Grouping Character	Description
( )	The parentheses characters serve to group terms and operators found between the characters
[ ]	The bracket characters serve to group terms and operators found between the characters; however, they prevent penetrations for the expansion operators (fuzzy, soundex, stem).

The beginning of a group of terms and operators is indicated by an open character from one of the sets of grouping characters. The ending of a group is indicated by the occurrence of the appropriate close character for the open character that started the group. Between the two characters, other groups may occur.

For example, the open parenthesis indicates the beginning of a group. The first close parenthesis encountered is the end of the group. Any open parentheses encountered before the close parenthesis indicate nested groups.

Escape Characters

To query on words or symbols that have special meaning to query expressions such as *and* & *or* | *accum*, you must escape them. There are two ways to escape characters in a query expression:

Table 4–2 Characters for Escaping Query Terms

Escape Character	Description
{}	Use braces to escape a string of characters or symbols. Everything within a set of braces in considered part of the escape sequence. When you use braces to escape a single character, the escaped character becomes a separate token in the query.
\	Use the backslash character to escape a single character or symbol. Only the character immediately following the backslash is escaped. For example, a query of *blue\-green* matches *blue-green* and *blue green*.

In the following examples, an escape sequence is necessary because each expression contains a Text operator or reserved symbol:

```
'high\-voltage'
'{high-voltage}'

'XY\&Z'
'{XY&Z}'
```

In the first example, the query matches *high-voltage* or *high voltage*.

Note that in the second example, a query on *XY&Z* will return 'XY Z', 'XY-Z', 'XY*Z', and so forth, as well as 'XY&Z'. This is because non-alphabetic characters are treated as whitespace (so *XY&Z* is treated as 'XY Z'). To match only *XY&Z*, you must declare *&* as a printjoin. (If you do, however, *XY&Z* will not match 'XY & Z'.) For more on printjoins, see BASIC_LEXER on page 2-31.

Note: If you use braces to escape an individual character within a word, the character is escaped, but the word is broken into three tokens.

For example, a query written as *high{-}voltage* searches for *high - voltage*, with the space on either side of the hyphen.

Querying Escape Characters

The open brace { signals the beginning of the escape sequence, and the closed brace } indicates the end of the sequence. Everything between the opening brace and the closing brace is part of the escaped query expression (including any open brace characters). To include the close brace character in an escaped query expression, use `}}`.

To escape the backslash escape character, use `\\`.

Reserved Words and Characters

Table 4–3 lists the Oracle Text reserved words and characters that must be escaped when you want to search them in `CONTAINS` queries:

Table 4–3 Reserved Words and Characters

Reserved Words	Reserved Characters	Operator
ABOUT	(none)	ABOUT

Table 4–3 (Cont.) Reserved Words and Characters

Reserved Words	Reserved Characters	Operator
ACCUM	,	Accumulate
AND	&	And
BT	(none)	Broader Term
BTG	(none)	Broader Term Generic
BTI	(none)	Broader Term Instance
BTP	(none)	Broader Term Partitive
EQUIV	=	Equivalence
FUZZY	?	fuzzy
(none)	{ }	escape characters (multiple)
(none)	\	escape character (single)
(none)	( )	grouping characters
(none)	[ ]	grouping characters
HASPATH	(none)	HASPATH
INPATH	(none)	INPATH
MDATA	(none)	MDATA
MINUS	-	MINUS
NEAR	;	NEAR
NOT	~	NOT
NT	(none)	Narrower Term
NTG	(none)	Narrower Term Generic
NTI	(none)	Narrower Term Instance
NTP	(none)	Narrower Term Partitive
OR	|	OR
PT	(none)	Preferred Term
RT	(none)	Related Term
(none)	$	stem
(none)	!	soundex
SQE	(none)	Stored Query Expression
SYN	(none)	Synonym
(none)	>	threshold
TR	(none)	Translation Term
TRSYN	(none)	Translation Term Synonym
TT	(none)	Top Term
(none)	*	weight
(none)	%	wildcard character (multiple)
(none)	_	wildcard character (single)

Table 4–3 (Cont.) Reserved Words and Characters

Reserved Words	Reserved Characters	Operator
WITHIN	(none)	WITHIN

5

CTX_ADM Package

This chapter provides information for using the CTX_ADM PL/SQL package.

CTX_ADM contains the following stored procedures:

Name	Description
MARK_FAILED	Changes an index's status from LOADING to FAILED.
RECOVER	Cleans up database objects for deleted Text tables.
SET_PARAMETER	Sets system-level defaults for index creation.

Note: Only the CTXSYS user can use the procedures in CTX_ADM.

MARK_FAILED

Use this procedure to change the status of an index from LOADING to FAILED.

Under rare circumstances, if CREATE INDEX or ALTER INDEX fails, an index may be left with the status LOADING. When an index is in LOADING status, any attempt to recover using RESUME INDEX is blocked. For this situation, use CTX_ADM.MARK_FAILED to forcibly change the status from LOADING to FAILED so that you can recover the index with RESUME INDEX.

You must log on as CTXSYS to run CTX_ADM.MARK_FAILED.

> **CAUTION: Use CTX_ADM.MARK_FAILED with caution. It should only be used as a last resort and only when no other session is touching the index. Normally, CTX_ADM.MARK_FAILED does not succeed if another session is actively building the index with CREATE or ALTER INDEX. However, index creation or alteration may include windows of time during which CTX_ADM.MARK_FAILED can succeed, marking the index as failed even as it is being built by another session.**

CTX_ADM.MARK_FAILED works with local partitioned indexes. However, it changes the status of all partitions to FAILED. Therefore, you should rebuild all index partitions with ALTER INDEX REBUILD PARTITION PARAMETERS ('RESUME') after using CTX_ADM.MARK_FAILED. If you run ALTER INDEX PARAMETER ('RESUME') after this operation, then Oracle resets the index partition status to valid. Oracle does not rebuild the index partitions that were successfully built before the MARK_FAILED operation.

Syntax

```
CTX_ADM.MARK_FAILED(
  owner_name     in    VARCHAR2,
  index_name     in    VARCHAR2);
```

owner_name
The name of the owner of the index whose status is to be changed.

index_name
The name of the index whose status is to be changed.

Example

```
begin
   CTX_ADM.MARK_FAILED('owner_1', 'index_1');
end;
```

RECOVER

The RECOVER procedure cleans up the Text data dictionary, deleting objects such as leftover preferences.

Syntax

```
CTX_ADM.RECOVER;
```

Example

```
begin
   ctx_adm.recover;
end;
```

SET_PARAMETER

The SET_PARAMETER procedure sets system-level parameters for index creation.

Syntax

```
CTX_ADM.SET_PARAMETER(param_name IN VARCHAR2,
                      param_value IN VARCHAR2);
```

param_name
Specify the name of the parameter to set, which can be one of the following parameters:

- max_index_memory (maximum memory allowed for indexing)
- default_index_memory (default memory allocated for indexing)
- log_directory (directory for CTX_OUPUT files)
- ctx_doc_key_type (default input key type for CTX_DOC procedures)
- file_access_role (default database role name for index creation when using FILE or URL datastores)
- default_datastore (default datastore preference)
- default_filter_file (default filter preference for data stored in files)
- default_filter_text (default text filter preference)
- default_filter_binary (default binary filter preference)
- default_section_html (default html section group preference)
- default_section_xml (default xml section group preference)
- default_section_text (default text section group preference)
- default_lexer (default lexer preference)
- default_wordlist (default wordlist preference)
- default_stoplist (default stoplist preference)
- default_storage (default storage preference)
- default_ctxcat_lexer
- default_ctxcat_stoplist
- default_ctxcat_storage
- default_ctxcat_wordlist
- default_ctxrule_lexer
- default_ctxrule_stoplist
- default_ctxrule_storage
- default_ctxrule_wordlist

> **See Also:** To learn more about the default values for these parameters, see "System Parameters" on page 2-76 in Chapter 2, "Oracle Text Indexing Elements"

param_value
Specify the value to assign to the parameter. For `max_index_memory` and `default_index_memory`, the value you specify must have the following syntax:

```
number[K|M|G]
```

where K stands for kilobytes, M stands for megabytes, and G stands for gigabytes.

For each of the other parameters, specify the name of a preference to use as the default for indexing.

Example

```
begin
  ctx_adm.set_parameter('default_lexer', 'my_lexer');
end;
```

6

CTX_CLS Package

This chapter provides reference information for using the `CTX_CLS` PL/SQL package, which enables you to perform document classification.

The following procedures are described in this chapter:

Name	Description
TRAIN	Generates rules that define document categories. Output based on input training document set.
CLUSTERING	Generates clusters for a document collection.

> **See Also:** *Oracle Text Application Developer's Guide* for more on document classification

TRAIN

Use this procedure to generate query rules that select document categories. You must supply a training set consisting of categorized documents. Documents can be in any format supported by Oracle Text and must belong to one or more categories. This procedure generates the queries that define the categories and then writes the results to a table.

You must also have a document table and a category table. The category table must contain at least two categories.

For example, your document and category tables can be defined as:

```
create table trainingdoc(
docid number primary key,
text varchar2(4000));

create table category (
docid trainingdoc(docid),
categoryid number);
```

You can use one of two syntaxes depending on the classification algorithm you need. The query compatible syntax uses the RULE_CLASSIFIER preference and generates rules as query strings. The support vector machine syntax uses the SVM_CLASSIFER preference and generates rules in binary format. The SVM_CLASSIFIER is good for high classification accuracy, but because its rules are generated in binary format, they cannot be examined like the query strings generated with the RULE_CLASSIFIER. Note that only those document ids that appear in both the document table and the category table will impact RULE_CLASSIFIER and SVM_CLASSIFIER learning.

The CTX_CLS.TRAIN procedure requires that your document table have an associated context index. For best results, the index should be synchronized before running this procedure. SVM_CLASSIFIER syntax enables the use of an unpopulated context index, while query-compatible syntax requires that the context index be populated.

> **See Also:** *Oracle Text Application Developer's Guide* for more on document classification

Query Compatible Syntax

The following syntax generates query-compatible rules and is used with the RULE_CLASSIFIER preference. Use this syntax and preference when different categories are separated from others by several key words. An advantage of generating your rules as query strings is that you can easily examine the generated rules. This is different from generating SVM rules, which are in binary format.

```
CTX_CLS.TRAIN(
index_name    in varchar2,
docid         in varchar2,
cattab        in varchar2,
catdocid      in varchar2,
catid         in varchar2,
restab        in varchar2,
rescatid      in varchar2,
resquery      in varchar2,
resconfid     in varchar2,
preference    in varchar2 DEFAULT NULL
);
```

index_name
Specify the name of the context index associated with your document training set.

docid
Specify the name of the document ID column in the document table. The document IDs in this column must be unique, and this column must be of datatype NUMBER. The values for this column must be stored in an unsigned 32-bit integer and must be in the range 0-4294967295.

cattab
Specify the name of the category table. You must have SELECT privilege on this table.

catdocid
Specify the name of the document ID column in the category table. The document ids in this table must also exist in the document table. This column must a NUMBER. The values for this column must be stored in an unsigned 32-bit integer and must be in the range 0-4294967295.

catid
Specify the name of the category ID column in the category table. This column must a NUMBER. The values for this column must be stored in an unsigned 32-bit integer and must be in the range 0-4294967295.

restab
Specify the name of the result table. You must have INSERT privilege on this table.

rescatid
Specify the name of the category ID column in the result table. This column must a NUMBER. The values for this column must be stored in an unsigned 32-bit integer and must be in the range 0-4294967295.

resquery
Specify the name of the query column in the result table. This column must be VARACHAR2, CHAR CLOB, NVARCHAR2, or NCHAR.

The queries generated in this column connects terms with AND or NOT operators, such as:

```
'T1 & T2 ~ T3'
```

Terms can also be theme tokens and be connected with the ABOUT operator, such as:

```
'about(T1) & about(T2) ~ about(T3)'
```

Generated rules also support WITHIN queries on field sections.

resconfid
Specify the name of the confidence column in result table. This column contains the estimated probability from training data that a document is relevant if that document satisfies the query.

preference
Specify the name of the preference. For classifier types and attributes, see "Classifier Types" on page 2-68 in Chapter 2, "Oracle Text Indexing Elements".

Syntax for Support Vector Machine Rules

The following syntax generates support vector machine (SVM) rules with the SVM_CLASSIFIER preference. This preference generates rules in binary format. Use this syntax when your application requires high classification accuracy.

```
CTX_CLS.TRAIN(
    index_name in varchar2,
    docid      in varchar2,
    cattab     in varchar2,
    catdocid   in varchar2,
    catid      in varchar2,
    restab     in varchar2,
    preference in varchar2 );
```

index_name
Specify the name of the text index.

docid
Specify the name of `docid` column in document table.

cattab
Specify the name of category table.

catdocid
Specify the name of `docid` column in category table.

catid
Specify the name of category ID column in category table.

restab
Specify the name of result table.

The result table has the following format:

Column Name	Datatype	Description
`CAT_ID`	`NUMBER`	The ID of the category.
`TYPE`	`NUMBER(3) NOT NULL`	0 for the actual rule or catid; 1 for other.
`RULE`	`BLOB`	The returned rule.

preference
Specify the name of user preference. For classifier types and attributes, see "Classifier Types" on page 2-68 in Chapter 2, "Oracle Text Indexing Elements".

Example

The `CTX_CLS.TRAIN` procedure is used in supervised classification. For an extended example, see *Oracle Text Application Developer's Guide*.

CLUSTERING

Use this procedure to cluster a collection of documents. A *cluster* is a group of documents similar to each other in content.

A clustering result set is composed of *document assignments* and *cluster descriptions*:

- A document assignment result set shows how relevant each document is to all generated leaf clusters.
- A cluster description result set contains information about what topic a cluster is about. This result set identifies the cluster and contains cluster description text, a suggested cluster label, and a quality score for the cluster.

Cluster output is hierarchical. Only leaf clusters are scored for relevance to documents. Producing more clusters requires more computing time. Indicate the upper limit for generated clusters with the `CLUSTER_NUM` attribute of the `KMEAN_CLUSTERING` cluster type (see "Cluster Types" on page 2-71 in this chapter).

There are two versions of this procedure: one with a table result set, and one with an in-memory result set.

Clustering is also known as *unsupervised classification*.

> **See Also:** For more information about clustering and relevant preferences, see Cluster Types on page 2-71 in Chapter 2, "Oracle Text Indexing Elements", as well as the *Oracle Text Application Developer's Guide*

Syntax: Table Result Set

```
ctx_cls.clustering (
 index_name  IN VARCHAR2,
 docid       IN VARCHAR2,
 doctab_name IN VARCHAR2,
 clstab_name IN VARCHAR2,
 pref_name   IN VARCHAR2  DEFAULT NULL
);
```

index_name
Specify the name of the context index on collection table.

docid
Specify the name of document ID column of the collection table.

doctab_name
Specify the name of document assignment table. This procedure creates the table with the following structure:

```
doc_assign(
   docid number,
   clusterid number,
   score number
);
```

Column	Description
DOCID	Document ID to identify document.

Column	Description
CLUSTERID	ID of a leaf cluster associated with this document. If CLUSTERID is -1, then the cluster contains "miscellaneous" documents; for example, documents that cannot be assigned to any other cluster category.
SCORE	The associated score between the document and the cluster.

If you require more columns, then create the table before you call this procedure.

clstab_name
Specify the name of the cluster description table. This procedure creates the table with the following structure:

```
cluster_desc(
  clusterid NUMBER,
  descript VARCHAR2(4000),
  label VARCHAR2(200),
  sze NUMBER,
  quality_score NUMBER,
  parent NUMBER
);
```

Column	Description
CLUSTERID	Cluster ID to identify cluster. If CLUSTERID is -1, then the cluster contains "miscellaneous" documents; for example, documents that cannot be assigned to any other cluster category.
DESCRIPT	String to describe the cluster.
LABEL	A suggested label for the cluster.
SZE	This parameter currently has no value.
QUALITY_SCORE	The quality score of the cluster. A higher number indicates greater coherence.
PARENT	The parent cluster ID. Zero means no parent cluster.

If you require more columns, then create the table before you call this procedure.

pref_name
Specify the name of the preference.

Syntax: In-Memory Result Set

Put the result set into in-memory structures for better performance. Two in-memory tables are defined in CTX_CLS package for document assignment and cluster description respectively.

```
CTX_CLS.CLUSTERING(
  index_name     IN VARCHAR2,
  docid          IN VARCHAR2,
  dids           IN DOCID_TAB,
  doctab_name    IN OUT NOCOPY DOC_TAB,
  clstab_name    IN OUT NOCOPY CLUSTER_TAB,
  pref_name      IN VARCHAR2  DEFAULT NULL
          );
```

index_name
Specify the name of context index on the collection table.

docid
Specify the document ID column of the collection table.

dids
Specify the name of the in-memory `docid_tab`.

```
TYPE docid_tab IS TABLE OF number INDEX BY BINARY_INTEGER;
```

doctab_name
Specify name of the document assignment in-memory table. This table is defined as follows:

```
TYPE doc_rec IS RECORD (
   docid NUMBER,
   clusterid NUMBER,
   score NUMBER
)
TYPE doc_tab IS TABLE OF doc_rec INDEX BY BINARY_INTEGER;
```

Column	Description
DOCID	Document ID to identify document.
CLUSTERID	ID of a leaf cluster associated with this document. If CLUSTERID is -1, then the cluster contains "miscellaneous" documents; for example, documents that cannot be assigned to any other cluster category.
SCORE	The associated score between the document and the cluster.

cls_tab
Specify the name of cluster description in-memory table.

```
TYPE cluster_rec IS RECORD(
     clusterid NUMBER,
     descript VARCHAR2(4000),
     label VARCHAR2(200),
     sze NUMBER,
     quality_score NUMBER,
     parent NUMBER
);
TYPE cluster_tab IS TABLE OF cluster_rec INDEX BY BINARY_INTEGER;
```

Column	Description
CLUSTERID	Cluster ID to identify cluster. If CLUSTERID is -1, then the cluster contains "miscellaneous" documents; for example, documents that cannot be assigned to any other cluster category.
DESCRIPT	String to describe the cluster.
LABEL	A suggested label for the cluster.
SZE	This parameter currently has no value.
QUALITY_SCORE	The quality score of the cluster. A higher number indicates greater coherence.
PARENT	The parent cluster ID. Zero means no parent cluster.

pref_name
Specify the name of the preference. For cluster types and attributes, see Cluster Types in Chapter 2, "Oracle Text Indexing Elements".

Example

See Also: The *Oracle Text Application Developer's Guide* for an example of using clustering

7

CTX_DDL Package

This chapter provides reference information for using the CTX_DDL PL/SQL package to create and manage the preferences, section groups, and stoplists required for Text indexes.

CTX_DDL contains the following stored procedures and functions:

Name	Description
ADD_ATTR_SECTION	Adds an attribute section to an XML section group.
ADD_FIELD_SECTION	Creates a field section and assigns it to the specified section group.
ADD_INDEX	Adds an index to a catalog index preference.
ADD_MDATA	Changes the MDATA value of a document.
ADD_MDATA_COLUMN	Maps a FILTER BY column to the specified MDATA section.
ADD_MDATA_SECTION	Adds an MDATA metadata section to a document.
ADD_NDATA_SECTION	Adds an NDATA section to a document.
ADD_SDATA_COLUMN	Maps a FILTER BY column to the specified SDATA section.
ADD_SDATA_SECTION	Adds an SDATA structured data section to a document.
ADD_SPECIAL_SECTION	Adds a special section to a section group.
ADD_STOPCLASS	Adds a stopclass to a stoplist.
ADD_STOP_SECTION	Adds a stop section to an automatic section group.
ADD_STOPTHEME	Adds a stoptheme to a stoplist.
ADD_STOPWORD	Adds a stopword to a stoplist.
ADD_SUB_LEXER	Adds a sub-lexer to a multi-lexer preference.
ADD_ZONE_SECTION	Creates a zone section and adds it to the specified section group.
COPY_POLICY	Creates a copy of a policy.
CREATE_INDEX_SET	Creates an index set for CTXCAT index types.
CREATE_POLICY	Creates a policy to use with ORA:CONTAINS().
CREATE_PREFERENCE	Creates a preference in the Text data dictionary.
CREATE_SECTION_GROUP	Creates a section group in the Text data dictionary.

Name	Description
CREATE_SHADOW_INDEX	Creates a policy for the passed-in index. For non-partitioned index, also creates an index table.
CREATE_STOPLIST	Creates a stoplist.
DROP_INDEX_SET	Drops an index set.
DROP_POLICY	Drops a policy.
DROP_PREFERENCE	Deletes a preference from the Text data dictionary.
DROP_SECTION_GROUP	Deletes a section group from the Text data dictionary.
DROP_SHADOW_INDEX	Drops a shadow index.
DROP_STOPLIST	Drops a stoplist.
EXCHANGE_SHADOW_INDEX	Swaps the shadow index metadata and data.
OPTIMIZE_INDEX	Optimizes the index.
POPULATE_PENDING	Populates the pending queue with every rowid in the base table or table partition.
RECREATE_INDEX_ONLINE	Recreates the passed-in index.
REMOVE_INDEX	Removes an index from a `CTXCAT` index preference.
REMOVE_MDATA	Removes `MDATA` values from a document.
REMOVE_SECTION	Deletes a section from a section group.
REMOVE_STOPCLASS	Deletes a stopclass from a stoplist.
REMOVE_STOPTHEME	Deletes a stoptheme from a stoplist.
REMOVE_STOPWORD	Deletes a stopword from a stoplist.
REMOVE_SUB_LEXER	Deletes a sub-lexer from a multi-lexer preference.
REPLACE_INDEX_METADATA	Replaces metadata for local domain indexes.
SET_ATTRIBUTE	Sets a preference attribute.
SYNC_INDEX	Synchronizes the index.
UNSET_ATTRIBUTE	Removes a set attribute from a preference.
UPDATE_POLICY	Updates a policy.

ADD_ATTR_SECTION

Adds an attribute section to an XML section group. This procedure is useful for defining attributes in XML documents as sections. This enables you to search XML attribute text with the WITHIN operator.

> **Note:** When you use AUTO_SECTION_GROUP, attribute sections are created automatically. Attribute sections created automatically are named in the form tag@attribute.

Syntax

```
CTX_DDL.ADD_ATTR_SECTION(
  group_name     in    varchar2,
  section_name   in    varchar2,
  tag            in    varchar2);
```

group_name
Specify the name of the XML section group. You can add attribute sections only to XML section groups.

section_name
Specify the name of the attribute section. This is the name used for WITHIN queries on the attribute text.

The section name you specify cannot contain the colon (:), comma (,), or dot (.) characters. The section name must also be unique within group_name. Section names are case-insensitive.

Attribute section names can be no more than 64 bytes long.

tag
Specify the name of the attribute in tag@attr form. This parameter is case-sensitive.

Examples

Consider an XML file that defines the BOOK tag with a TITLE attribute as follows:

```
<BOOK TITLE="Tale of Two Cities">
  It was the best of times.
</BOOK>
```

To define the title attribute as an attribute section, create an XML_SECTION_GROUP and define the attribute section as follows:

```
begin
ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'BOOK@TITLE');
end;
```

When you define the TITLE attribute section as such and index the document set, you can query the XML attribute text as follows:

```
'Cities within booktitle'
```

ADD_FIELD_SECTION

Creates a field section and adds the section to an existing section group. This enables field section searching with the WITHIN operator.

Field sections are delimited by start and end tags. By default, the text within field sections are indexed as a sub-document separate from the rest of the document.

Unlike zone sections, field sections cannot nest or overlap. As such, field sections are best suited for non-repeating, non-overlapping sections such as `TITLE` and `AUTHOR` markup in e-mail- or news-type documents.

Because of how field sections are indexed, WITHIN queries on field sections are usually faster than `WITHIN` queries on zone sections.

Syntax

```
CTX_DDL.ADD_FIELD_SECTION(
  group_name     in    varchar2,
  section_name   in    varchar2,
  tag            in    varchar2,
  visible        in    boolean default FALSE
);
```

group_name
Specify the name of the section group to which `section_name` is added. You can add up to 64 field sections to a single section group. Within the same group, section zone names and section field names cannot be the same.

section_name
Specify the name of the section to add to the `group_name`. Use this name to identify the section in queries. Avoid using names that contain non-alphanumeric characters such as _, because these characters must be escaped in queries. Section names are case-insensitive.

Within the same group, zone section names and field section names cannot be the same. The terms *Paragraph* and *Sentence* are reserved for special sections.

Section names need not be unique across tags. You can assign the same section name to more than one tag, which makes details transparent to searches.

tag
Specify the tag that marks the start of a section. For example, if the tag is <H1>, then specify H1. The start tag you specify must be unique within a section group.

If `group_name` is an `HTML_SECTION_GROUP`, then you can create field sections for the META tag's `NAME/CONTENT` attribute pairs. To do so, specify `tag` as `meta@namevalue` where `namevalue` is the value of the `NAME` attribute whose `CONTENT` attribute is to be indexed as a section. Refer to the example "Creating Sections for <META> Tags" on page 7-5.

Oracle Text knows what the end tags look like from the `group_type` parameter you specify when you create the section group.

visible
Specify `TRUE` to make the text visible within the rest of the document.

By default the `visible` flag is `FALSE`. This means that Oracle Text indexes the text within field sections as a sub-document separate from the rest of the document.

However, you can set the visible flag to TRUE if you want text within the field section to be indexed as part of the enclosing document.

Examples

Visible and Invisible Field Sections

The following example defines a section group basicgroup of the BASIC_SECTION_GROUP type. It then creates a field section in basicgroup called Author for the <A> tag. It also sets the visible flag to FALSE:

```
begin
ctx_ddl.create_section_group('basicgroup', 'BASIC_SECTION_GROUP');
ctx_ddl.add_field_section('basicgroup', 'Author', 'A', FALSE);
end;
```

Because the Author field section is not visible, to find text within the Author section, you must use the WITHIN operator as follows:

```
'(Martin Luther King) WITHIN Author'
```

A query of *Martin Luther King* without the WITHIN operator does not return instances of this term in field sections. To query text within field sections without specifying WITHIN, you must set the visible flag to TRUE when you create the section as follows:

```
begin
ctx_ddl.add_field_section('basicgroup', 'Author', 'A', TRUE);
end;
```

Creating Sections for <META> Tags

When you use the HTML_SECTION_GROUP, you can create sections for META tags.

Consider an HTML document that has a META tag as follows:

```
<META NAME="author" CONTENT="ken">
```

To create a field section that indexes the CONTENT attribute for the <META NAME="author"> tag:

```
begin
ctx_ddl.create_section_group('myhtmlgroup', 'HTML_SECTION_GROUP');
ctx_ddl.add_field_section('myhtmlgroup', 'author', 'META@AUTHOR');
end
```

After indexing with section group mygroup, query the document as follows:

```
'ken WITHIN author'
```

Limitations

Nested Sections

Field sections cannot be nested. For example, if you define a field section to start with <TITLE> and define another field section to start with <FOO>, the two sections *cannot* be nested as follows:

```
<TITLE> dog <FOO> cat </FOO> </TITLE>
```

To work with nested section define them as zone sections.

Repeated Sections

Repeated field sections are allowed, but `WITHIN` queries treat them as a single section. The following is an example of repeated field section in a document:

```
<TITLE> cat </TITLE>
<TITLE> dog </TITLE>
```

The query (*dog and cat) within title* returns the document, even though these words occur in different sections.

To have `WITHIN` queries distinguish repeated sections, define them as zone sections.

Related Topics

"WITHIN" on page 3-60

"Section Group Types" on page 2-66

"CREATE_SECTION_GROUP" on page 7-38

"ADD_ZONE_SECTION" on page 7-28

"ADD_SPECIAL_SECTION" on page 7-18

"REMOVE_SECTION" on page 7-67

"DROP_SECTION_GROUP" on page 7-48

ADD_INDEX

Use this procedure to add a sub-index to a catalog index preference. Create this preference by naming one or more columns in the base table.

Because you create sub-indexes to improve the response time of structured queries, the column you add should be used in the `structured_query` clause of the `CATSEARCH` operator at query time.

Syntax

```
CTX_DDL.ADD_INDEX(set_name in varchar2,
column_list varchar2,
storage_clause varchar2);
```

set_name
Specify the name of the index set.

column_list
Specify a comma separated list of columns to index. At index time, any column listed here cannot have a NULL value in any row in the base table. If any row is NULL during indexing, then an error is raised.

Always ensure that your columns have non-NULL values before and after indexing.

storage_clause
Specify a storage clause.

Example

Consider a table called `AUCTION` with the following schema:

```
create table auction(
item_id number,
title varchar2(100),
category_id number,
price number,
bid_close date);
```

Assume that queries on the table involve a mandatory text query clause and optional structured conditions on `category_id`. Results must be sorted based on `bid_close`.

You can create a catalog index to support the different types of structured queries a user might enter.

To create the indexes, first create the index set preference then add the required indexes to it:

```
begin
  ctx_ddl.create_index_set('auction_iset');
  ctx_ddl.add_index('auction_iset','bid_close');
  ctx_ddl.add_index('auction_iset','category_id, bid_close');
end;
```

Create the combined catalog index with `CREATE INDEX` as follows:

```
create index auction_titlex on AUCTION(title) indextype is CTXCAT parameters
('index set auction_iset');
```

Querying

To query the title column for the word *pokemon*, enter regular and mixed queries as follows:

```
select * from AUCTION where CATSEARCH(title, 'pokemon',NULL)> 0;
select * from AUCTION where CATSEARCH(title, 'pokemon', 'category_id=99 order by
bid_close desc')> 0;
```

Notes

VARCHAR2 columns in the column list of a CTXCAT index of an index set cannot exceed 30 bytes.

Related Topic

"REMOVE_INDEX" on page 7-65

ADD_MDATA

Use this procedure to change the metadata of a document that has been specified as an MDATA section. After this call, MDATA queries involving the named MDATA value will find documents with the given MDATA value.

There are two versions of CTX_DDL.ADD_MDATA: one for adding a single metadata value to a single rowid, and one for handing multiple values, multiple rowids, or both.

CTX_DDL.ADD_MDATA is transactional; it takes effect immediately in the calling session, can be seen only in the calling session, can be reversed with a ROLLBACK command, and must be committed to take permanent effect.

Use CTX_DDL.REMOVE_MDATA to remove metadata values from already-indexed documents. Only the owner of the index is allowed to call ADD_MDATA and REMOVE_MDATA.

Syntax

This is the syntax for adding a single value to a single rowid:

```
CTX_DDL.ADD_MDATA(
     idx_name           IN VARCHAR2,
     section_name       IN VARCHAR2,
     mdata_value        IN VARCHAR2,
     mdata_rowid        IN VARCHAR2,
     [part_name]        IN VARCHAR2]
);
```

idx_name
Name of the text index that contains the named *rowid*.

section_name
Name of the MDATA section.

mdata_value
The metadata value to add to the document.

mdata_rowid
The rowid to which to add the metadata value.

[part_name]
Name of the index partition, if any. Must be provided for local partitioned indexes and must be NULL for global, non-partitioned indexes.

This is the syntax for handling multiple values, multiple rowids, or both. This version is more efficient for large numbers of new values or rowids.

```
CTX_DDL.ADD_MDATA(
     idx_name           IN VARCHAR2,
     section_name       IN VARCHAR2,
     mdata_values       SYS.ODCIVARCHAR2LIST,
     mdata_rowids       SYS.ODCIRIDLIST,
     [part_name]        IN VARCHAR2]
);
```

idx_name
Name of the text index that contains the named *rowids*.

section_name
Name of the MDATA section.

mdata_values
List of metadata values. If a metadata value contains a comma, the comma must be escaped with a backslash.

mdata_rowids
The rowids to which to add the metadata values.

[part_name]
Name of the index partition, if any. Must be provided for local partitioned indexes and must be NULL for global, non-partitioned indexes.

Example

This example updates a single value:

```
select rowid from mytab where contains(text, 'MDATA(sec, value')>0;
No rows returned
exec ctx_ddl.add_mdata('my_index', 'sec', 'value', 'ABC');
select rowid from mytab where contains(text, 'MDATA(sec, value')>0;
ROWID
-----
ABC
```

This example updates multiple values:

```
begin
ctx_ddl.add_mdata('my_index', 'sec',
     sys.odcivarchar2list('value1','value2','value3'),
     sys.odciridlist('ABC','DEF'));
end;
```

This is equivalent to:

```
begin
ctx_ddl.add_mdata('my_index', 'sec', 'value1', 'ABC');
ctx_ddl.add_mdata('my_index', 'sec', 'value1', 'DEF');
ctx_ddl.add_mdata('my_index', 'sec', 'value2', 'ABC');
ctx_ddl.add_mdata('my_index', 'sec', 'value2', 'DEF');
ctx_ddl.add_mdata('my_index', 'sec', 'value3', 'ABC');
ctx_ddl.add_mdata('my_index', 'sec', 'value3', 'DEF');
end;
```

Notes

If a rowid is not yet indexed, CTX_DDL.ADD.MDATA completes without error, but an error is logged in CTX_USER_INDEX_ERRORS.

These updates are updates directly on the index itself, not on the actual contents stored in the base table. Therefore, they will not survive when the Text index is rebuilt.

Related Topics

See also "ADD_MDATA_SECTION" on page 7-12; "REMOVE_MDATA" on page 7-66; "MDATA" on page 3-28; as well as the Section Searching chapter of the *Oracle Text Application Developer's Guide*.

ADD_MDATA_COLUMN

Use this procedure to map the FILTER BY column named in *column_name* to the MDATA section named in *section_name*.

Syntax

The syntax is as follows:

```
CTX_DDL.ADD_MDATA_COLUMN(
     group_name         IN VARCHAR2,
     section_name       IN VARCHAR2,
     column_name        IN VARCHAR2,
);
```

group_name
Name of the group that contains the section.

section_name
Name of the MDATA section.

column_name
Name of the FILTER BY column to add to the MDATA section.

Restrictions

MDATA sections that are created with CTX_DDL.ADD_MDATA_COLUMN cannot have their values changed using CTX_DDL.ADD_MDATA or CTX_DDL.REMOVE_MDATA. Doing so will result in errors being returned. The section values must be updated using SQL.

Notes

- The stored datatype for MDATA sections is text. Therefore, the value of the FILTER BY column is converted to text during indexing. For non-text datatypes, the FILTER BY columns are normalized to an internal format during indexing. If the section is queried with an MDATA operator, then the MDATA query string will also be normalized to the internal format before processing.
- When a FILTER BY column is mapped as MDATA, the cost-based optimizer in Oracle Text tries to avoid using the Oracle Text composite domain index to process range predicate(s) on that FILTER BY column. This is because range predicates on MDATA FILTER BY columns are processed less efficiently than if they were declared as SDATA. For this reason, you should not add a FILTER BY column as MDATA if you plan to do range searches on the column.

Related Topics

"MDATA" on page 3-28

"ADD_MDATA_SECTION" on page 7-12

"REMOVE_MDATA" on page 7-66

"ADD_SDATA_COLUMN" on page 7-14

See Also: Chapter 8, "Searching Document Sections in Oracle Text" in *Oracle Text Application Developer's Guide*

ADD_MDATA_SECTION

Use this procedure to add an MDATA section, with an accompanying value, to an existing section group. MDATA sections cannot be added to Null Section groups, Path Section groups, or Auto Section groups.

Section values undergo a simplified normalization:

- Leading and trailing whitespace on the value is removed.
- The value is truncated to 64 bytes.
- The value is indexed as a single value; if the value consists of multiple words, it is not broken up.
- Case is preserved. If the document is dynamically generated, then implement case-insensitivity by uppercasing MDATA values and making sure to search only in uppercase.

Use CTX_DDL.REMOVE_SECTION to remove sections.

Syntax

```
CTX_DDL.ADD_MDATA_SECTION(
     group_name    IN VARCHAR2,
     section_name  IN VARCHAR2,
     tag           IN VARCHAR2,
);
```

group_name
Name of the section group that will contain the MDATA section.

section_name
Name of the MDATA section.

tag
The value of the MDATA section. For example, if the section is <AUTHOR>, the value could be *Cynthia Kadohata* (author of the novel *The Floating World*). More than one *tag* can be assigned to a given MDATA section.

Example

This example creates an MDATA section called auth.

```
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
ctx_ddl.add_mdata_section('htmgroup', 'auth', 'author');
```

Related Topics

"ADD_MDATA" on page 7-9

"REMOVE_MDATA" on page 7-66

"MDATA" on page 3-28

"CREATE_SECTION_GROUP" on page 7-38

The Section Searching chapter of the *Oracle Text Application Developer's Guide*

ADD_NDATA_SECTION

Use this procedure to find matches that are spelled in a similar way. The value of an NDATA section is extracted from the document text like other sections, but is indexed as name data. NDATA sections are stored in the CTX_USER_SECTIONS view.

Syntax

```
CTX_DDL.ADD_NDATA_SECTION (
   group_name    IN VARCHAR2,
   section_name  IN VARCHAR2,
   tag           IN VARCHAR2);
```

group_name
Name of the group that contains the section.

section_name
Name of the NDATA section.

tag
Name of the tag that marks the start of a section. For example, if the tag is <H1>, specify H1. The start tag you specify must be unique within a section group.

Notes

NDATA sections support both single and multi-byte data, however, there are character- and term-based limitations. NDATA section data that is indexed is constrained as follows:

- number of characters in a single, white space delimited term

  511

- number of white space delimited terms

  255

- total number of characters, including white spaces

  511

NDATA section data that exceeds these constraints are truncated.

Example

The following example defines a section group namegroup of the BASIC_SECTION_GROUP type. It then creates an NDATA section in namegroup called firstname.

```
begin
  ctx_ddl.create_section_group('namegroup', 'BASIC_SECTION_GROUP');
  ctx_ddl.add_ndata_section('namegroup', 'firstname', 'fname1');
end;
```

ADD_SDATA_COLUMN

Use this procedure to map the FILTER BY or ORDER BY column named in *column_name* to the SDATA section named in *section_name*. By default, all FILTER BY columns are mapped as SDATA.

Syntax

The syntax is as follows:

```
CTX_DDL.ADD_SDATA_COLUMN(
     group_name          IN VARCHAR2,
     section_name        IN VARCHAR2,
     column_name         IN VARCHAR2,
);
```

group_name
Name of the group that contains the section.

section_name
Name of the SDATA section.

column_name
Name of the FILTER BY column to add to the SDATA section.

Notes

- Mapping FILTER BY columns to sections is optional. If no section mapping exists for a FILTER BY column, then it is mapped to an SDATA section, and the section name will be the name of the FILTER BY column.
- If a section group is not specified during CREATE INDEX of a composite domain index, then system default section group settings will be used, and a SDATA section will be created for each of the FILTER BY and ORDER BY columns.

> **Note:** Because section name does not allow certain special characters and is case insensitive, if the column name is case sensitive or contains special characters, then an error will be raised. To work around this problem, you need to map the column to an MDATA or SDATA section before creating the index. Refer to CTX_DDL.ADD_MDATA_COLUMN or CTX_DDL.ADD_SDATA_COLUMN in this chapter.

- An error will be raised if a column mapped to MDATA also appears in the ORDER BY column clause.
- Column section names are unique to their section group. That is, you cannot have an MDATA column section named FOO if you already have an MDATA column section named FOO. Furthermore, you cannot have a field section named FOO if you already have an SDATA column section named FOO. This is true whether it is implicitly created (by CREATE INDEX for FILTER BY or ORDER BY clauses) or explicitly created (by CTX_DDL.ADD_SDATA_COLUMN).
- One section name can only be mapped to one FILTER BY column, and vice versa. For example, mapping a section to more than one column or mapping a column to more than one section is not allowed.

- Column sections can be added to any type of section group, including the `NULL` section group.
- 32 is the maximum number for `SDATA` sections and columns.

Related Topics

"SDATA" on page 3-44

"ADD_SDATA_SECTION" on page 7-16

See Also: Chapter 8, "Searching Document Sections in Oracle Text" in *Oracle Text Application Developer's Guide*

ADD_SDATA_SECTION

This procedure adds an SDATA section to a section group. By default, all FILTER BY columns are mapped as SDATA.

Syntax

The syntax is as follows:

```
CTX_DDL.ADD_SDATA_SECTION(
     group_name         IN VARCHAR2,
     section_name       IN VARCHAR2,
     tag                IN VARCHAR2,
     datatype           IN VARCHAR2, default NULL,
);
```

group_name
Name of the group that contains the section.

section_name
Name of the SDATA section.

tag
Name of the tag to add to the SDATA section.

datatype
Specifies the stored format for the data, as well as the semantics of comparison in later use in SDATA operators. The default is VARCHAR2, but if specified must be one of the following values:

- VARCHAR2
- CHAR
- RAW
- NUMBER
- DATE

The VARCHAR2 datatype stores up to 249 bytes of character data in the database character set. Values larger than this result in a per-document indexing error. Note that leading and trailing whitespace are always trimmed from SDATA section values when extracted by the sectioner. This is different than SDATA columns. Column values are never trimmed. No lexing is performed on the value from either kind of SDATA.

The CHAR datatype stores up to 249 bytes of character data in the database character set. Values larger than this result in a per-document indexing error. Note that leading and trailing whitespace are always trimmed from SDATA section values when extracted by the sectioner. This is different than SDATA columns. Column values are never trimmed. No lexing is performed on the value from either kind of SDATA. To be consistent with SQL, the comparisons of CHAR datatype SDATA values are blank-padded comparisons.

RAW datatype stores up to 249 bytes of binary data. Values larger than this result in a per-document indexing error. The value is converted from hexadecimal string representation. That is, to store a value of 65, the document should look like <TAG>40</TAG>, and not <TAG>65</TAG> or <TAG>A</TAG>.

The DATE datatype values must conform to the following format: YYYY-MM-DD or YYYY-MM-DD HH24:MI:SS. That is, to store a DATE value of "Nov. 24, 2006 10:32pm 36sec", the document should look like <TAG>2006-11-24 22:32:36</TAG>.

Limitations

- SDATA are single-occurrence only. If multiple instances of an SDATA tag are encountered in a single document, then later instances supersede the value set by earlier instances. This means that the last occurrence of an SDATA tag takes effect.
- If no SDATA tag occurs in a given document, then this is treated as an SDATA value of NULL.
- Empty SDATA tags are treated as NULL values.
- SDATA sections cannot be nested. Sections that are nested inside are ignored.
- 32 is the maximum number for SDATA sections and columns.

Related Topics

"SDATA" on page 3-44

"ADD_SDATA_COLUMN" on page 7-14

See Also: Chapter 8, "Searching Document Sections in Oracle Text" in *Oracle Text Application Developer's Guide*

ADD_SPECIAL_SECTION

Adds a special section, either SENTENCE or PARAGRAPH, to a section group. This enables searching within sentences or paragraphs in documents with the WITHIN operator.

A special section in a document is a section which is not explicitly tagged like zone and field sections. The start and end of special sections are detected when the index is created. Oracle Text supports two such sections: *paragraph* and *sentence*.

The sentence and paragraph boundaries are determined by the lexer. For example, the lexer recognizes sentence and paragraph section boundaries as follows:

Table 7–1 Paragraph and Sentence Section Boundaries

Special Section	Boundary
SENTENCE	WORD/PUNCT/WHITESPACE
	WORD/PUNCT/NEWLINE
PARAGRAPH	WORD/PUNCT/NEWLINE/WHITESPACE (indented paragraph)
	WORD/PUNCT/NEWLINE/NEWLINE (block paragraph)

The punctuation, whitespace, and newline characters are determined by your lexer settings and can be changed.

If the lexer cannot recognize the boundaries, no sentence or paragraph sections are indexed.

Syntax

```
CTX_DDL.ADD_SPECIAL_SECTION(
                 group_name    IN VARCHAR2,
                 section_name  IN VARCHAR2);
```

group_name
Specify the name of the section group.

section_name
Specify SENTENCE or PARAGRAPH.

Example

The following example enables searching within sentences within HTML documents:

```
begin
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
ctx_ddl.add_special_section('htmgroup', 'SENTENCE');
end;
```

Add zone sections to the group to enable zone searching in addition to sentence searching. The following example adds the zone section Headline to the section group htmgroup:

```
begin
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
ctx_ddl.add_special_section('htmgroup', 'SENTENCE');
ctx_ddl.add_zone_section('htmgroup', 'Headline', 'H1');
end;
```

If you are only interested in sentence or paragraph searching within documents and not interested in defining zone or field sections, then use the NULL_SECTION_GROUP as follows:

```
begin
ctx_ddl.create_section_group('nullgroup', 'NULL_SECTION_GROUP');
ctx_ddl.add_special_section('nullgroup', 'SENTENCE');
end;
```

Related Topics

"WITHIN" on page 3-60

"Section Group Types" on page 2-66

"CREATE_SECTION_GROUP" on page 7-38

"ADD_ZONE_SECTION" on page 7-28

"ADD_FIELD_SECTION" on page 7-4

"REMOVE_SECTION" on page 7-67

"DROP_SECTION_GROUP" on page 7-48

ADD_STOPCLASS

Adds a stopclass to a stoplist. A stopclass is a class of tokens that is not to be indexed.

Syntax

```
CTX_DDL.ADD_STOPCLASS(
  stoplist_name  in   varchar2,
  stopclass      in   varchar2
);
```

stoplist_name
Specify the name of the stoplist.

stopclass
Specify the stopclass to be added to `stoplist_name`. Currently, only the `NUMBERS` class is supported. It is not possible to create a custom stopclass.

`NUMBERS` includes tokens that follow the number pattern: digits, `numgroup`, and `numjoin` only. Therefore, *123ABC* is not a number, nor is *A123*. These are labeled as `MIXED`. *$123* is not a number (this token is not common in a text index because non-alphanumerics become whitespace by default). In the United States, *123.45* is a number, but *123.456.789* is not; in Europe, where numgroup may be '.', the reverse is true.

The maximum number of stopwords, stopthemes, and stopclasses you can add to a stoplist is 4095.

Example

The following example adds a stopclass of `NUMBERS` to the stoplist `mystop`:

```
begin
ctx_ddl.add_stopclass('mystop', 'NUMBERS');
end;
```

Related Topics

"CREATE_STOPLIST" on page 7-43

"REMOVE_STOPCLASS" on page 7-68

"DROP_STOPLIST" on page 7-50

ADD_STOP_SECTION

Adds a stop section to an automatic section group. Adding a stop section causes the automatic section indexing operation to ignore the specified section in XML documents.

> **Note:** Adding a stop section causes no section information to be created in the index. However, the text within a stop section is always searchable.

Adding a stop section is useful when your documents contain many low information tags. Adding stop sections also improves indexing performance with the automatic section group.

The number of stop sections you can add is unlimited.

Stop sections do not have section names and hence are not recorded in the section views.

Syntax

```
CTX_DDL.ADD_STOP_SECTION(
                section_group IN VARCHAR2,
                tag  IN VARCHAR2);
```

section_group
Specify the name of the automatic section group. If you do not specify an automatic section group, then this procedure returns an error.

tag
Specify the tag to ignore during indexing. This parameter is case-sensitive. Defining a stop tag as such also stops the tag's attribute sections, if any.

Qualify the tag with document type in the form `(doctype)tag`. For example, if you wanted to make the `<fluff>` tag a stop section only within the `mydoc` document type, specify `(mydoc)fluff` for tag.

Example

Defining Stop Sections

The following example adds a stop section identified by the tag `<fluff>` to the automatic section group `myauto`:

```
begin
ctx_ddl.add_stop_section('myauto', 'fluff');
end;
```

This example also stops any attribute sections contained within `<fluff>`. For example, if a document contained:

```
<fluff type="computer">
```

Then the preceding example also stops the attribute section `fluff@type`.

Doctype Sensitive Stop Sections

The following example creates a stop section for the tag <fluff> only in documents that have a root element of mydoc:

```
begin
ctx_ddl.add_stop_section('myauto', '(mydoc)fluff');
end;
```

Related Topics

"ALTER INDEX" on page 1-2

"CREATE_SECTION_GROUP" on page 7-38

ADD_STOPTHEME

Adds a single stoptheme to a stoplist. A stoptheme is a theme that is not to be indexed.

In English, query on indexed themes using the ABOUT operator.

Syntax

```
CTX_DDL.ADD_STOPTHEME(
  stoplist_name  in   varchar2,
  stoptheme      in   varchar2
);
```

stoplist_name
Specify the name of the stoplist.

stoptheme
Specify the stoptheme to be added to `stoplist_name`. The system normalizes the `stoptheme` you enter using the knowledge base. If the normalized theme is more than one theme, then the system does not process your stoptheme. For this reason, Oracle recommends that you submit single stopthemes.

The maximum number of stopwords, stopthemes, and stopclasses you can add to a stoplist is 4095.

Example

The following example adds the stoptheme `banking` to the stoplist `mystop`:

```
begin
ctx_ddl.add_stoptheme('mystop', 'banking');
end;
```

Related Topics

"CREATE_STOPLIST" on page 7-43

"REMOVE_STOPTHEME" on page 7-69

"DROP_STOPLIST" on page 7-50

"ABOUT" on page 3-4

ADD_STOPWORD

Use this procedure to add a single stopword to a stoplist.

To create a list of stopwords, you must call this procedure once for each word.

Syntax

```
CTX_DDL.ADD_STOPWORD(
stoplist_name  in   varchar2,
stopword       in   varchar2,
language       in varchar2 default NULL
);
```

stoplist_name
Specify the name of the stoplist.

stopword
Specify the stopword to be added.

Language-specific stopwords must be unique across the other stopwords specific to the language. For example, it is valid to have a German *die* and an English *die* in the same stoplist.

The maximum number of stopwords, stopthemes, and stopclasses you can add to a stoplist is 4095.

language
Specify the language of `stopword` when the stoplist you specify with `stoplist_name` is of type `MULTI_STOPLIST`. You must specify the globalization support name or abbreviation of an Oracle Text-supported language.

To make a stopword active in multiple languages, specify `ALL` for this parameter. For example, defining `ALL` stopwords is useful when you have international documents that contain English fragments that need to be stopped in any language.

An `ALL` stopword is active in all languages. If you use the multi-lexer, the language-specific lexing of the stopword occurs, just as if it had been added multiple times in multiple specific languages.

Otherwise, specify `NULL`.

Example

Single Language Stoplist

The following example adds the stopwords *because*, *notwithstanding*, *nonetheless*, and *therefore* to the stoplist `mystop`:

```
begin
ctx_ddl.add_stopword('mystop', 'because');
ctx_ddl.add_stopword('mystop', 'notwithstanding');
ctx_ddl.add_stopword('mystop', 'nonetheless');
ctx_ddl.add_stopword('mystop', 'therefore');
end;
```

Multi-Language Stoplist

The following example adds the German word *die* to a multi-language stoplist:

```
begin
ctx_ddl.add_stopword('mystop', 'Die','german');
end;
```

Note: Add stopwords after you create the index with `ALTER INDEX`.

Adding An ALL Stopword

The following adds the word *the* as an `ALL` stopword to the multi-language stoplist *globallist*:

```
begin
ctx_ddl.add_stopword('globallist','the','ALL');
end;
```

Related Topics

"CREATE_STOPLIST" on page 7-43

"REMOVE_STOPWORD" on page 7-70

"DROP_STOPLIST" on page 7-50

"ALTER INDEX" on page 1-2

Appendix E, "Oracle Text Supplied Stoplists"

ADD_SUB_LEXER

Add a sub-lexer to a multi-lexer preference. A sub-lexer identifies a language in a multi-lexer (multi-language) preference. Use a multi-lexer preference when you want to index more than one language.

Restrictions

The following restrictions apply to using CTX_DDL.ADD_SUB_LEXER:

- The invoking user must be the owner of the multi-lexer or CTXSYS.
- The lexer_name parameter must name a preference which is a multi-lexer lexer.
- A lexer for default must be defined before the multi-lexer can be used in an index.
- The sub-lexer preference owner must be the same as multi-lexer preference owner.
- The sub-lexer preference must not be a multi-lexer lexer.
- A sub-lexer preference cannot be dropped while it is being used in a multi-lexer preference.
- CTX_DDL.ADD_SUB_LEXER records only a reference. The sub-lexer values are copied at create index time to index value storage.

Syntax

```
CTX_DDL.ADD_SUB_LEXER(
             lexer_name in varchar2,
             language  in varchar2,
             sub_lexer in varchar2,
             alt_value in varchar2 default null
);
```

lexer_name
Specify the name of the multi-lexer preference.

language
Specify the globalization support language name or abbreviation of the sub-lexer. For example, specify JAPANESE or JA for Japanese.

The sub-lexer you specify with sub_lexer is used when the language column has a value case-insensitive equal to the globalization support name of abbreviation of language.

Specify DEFAULT to assign a default sub-lexer to use when the value of the language column in the base table is null, invalid, or unmapped to a sub-lexer. The DEFAULT lexer is also used to parse stopwords.

If a sub-lexer definition for language already exists, then it is replaced by this call.

sub_lexer
Specify the name of the sub-lexer to use for this language.

alt_value
Optionally specify an alternate value for language.

If you specify DEFAULT for language, then you cannot specify an alt_value.

The `alt_value` is limited to 30 bytes and cannot be a globalization support language name, abbreviation, or `DEFAULT`.

Example

This example shows how to create a multi-language text table and how to set up the multi-lexer to index the table.

Create the multi-language table with a primary key, a text column, and a language column as follows:

```
create table globaldoc (
   doc_id number primary key,
   lang varchar2(3),
   text clob
);
```

Assume that the table holds mostly English documents, with an occasional German or Japanese document. To handle the three languages, you must create three sub-lexers: one for English, one for German, and one for Japanese as follows:

```
ctx_ddl.create_preference('english_lexer','basic_lexer');
ctx_ddl.set_attribute('english_lexer','index_themes','yes');
ctx_ddl.set_attribtue('english_lexer','theme_language','english');

ctx_ddl.create_preference('german_lexer','basic_lexer');
ctx_ddl.set_attribute('german_lexer','composite','german');
ctx_ddl.set_attribute('german_lexer','mixed_case','yes');
ctx_ddl.set_attribute('german_lexer','alternate_spelling','german');

ctx_ddl.create_preference('japanese_lexer','japanese_vgram_lexer');
```

Create the multi-lexer preference:

```
ctx_ddl.create_preference('global_lexer', 'multi_lexer');
```

Because the stored documents are mostly English, make the English lexer the default:

```
ctx_ddl.add_sub_lexer('global_lexer','default','english_lexer');
```

Add the German and Japanese lexers in their respective languages. Also assume that the language column is expressed in ISO 639-2, so add those as alternative values.

```
ctx_ddl.add_sub_lexer('global_lexer','german','german_lexer','ger');
ctx_ddl.add_sub_lexer('global_lexer','japanese','japanese_lexer','jpn');
```

Create the index `globalx`, specifying the multi-lexer preference and the language column in the parameters string as follows:

```
create index globalx on globaldoc(text) indextype is ctxsys.context
parameters ('lexer global_lexer language column lang');
```

ADD_ZONE_SECTION

Creates a zone section and adds the section to an existing section group. This enables zone section searching with the WITHIN operator.

Zone sections are sections delimited by start and end tags. The <B> and </B> tags in HTML, for instance, marks a range of words which are to be rendered in boldface.

Zone sections can be nested within one another, can overlap, and can occur more than once in a document.

Syntax

```
CTX_DDL.ADD_ZONE_SECTION(
  group_name     in    varchar2,
  section_name   in    varchar2,
  tag            in    varchar2
);
```

group_name
Specify the name of the section group to which section_name is added.

section_name
Specify the name of the section to add to the group_name. Use this name to identify the section in WITHIN queries. Avoid using names that contain non-alphanumeric characters such as _, because most of these characters are special must be escaped in queries. Section names are case-insensitive.

Within the same group, zone section names and field section names cannot be the same. The terms *Paragraph* and *Sentence* are reserved for special sections.

Section names need not be unique across tags. You can assign the same section name to more than one tag, making details transparent to searches.

tag
Specify the pattern which marks the start of a section. For example, if <H1> is the HTML tag, specify H1 for tag. The start tag you specify must be unique within a section group.

Oracle Text knows what the end tags look like from the group_type parameter you specify when you create the section group.

If group_name is an HTML_SECTION_GROUP, you can create zone sections for the META tag's NAME/CONTENT attribute pairs. To do so, specify tag as meta@namevalue where namevalue is the value of the NAME attribute whose CONTENT attributes are to be indexed as a section. Refer to the example.

If group_name is an XML_SECTION_GROUP, you can optionally qualify tag with a document type (root element) in the form (doctype)tag. Doing so makes section_name sensitive to the XML document type declaration. Refer to the example.

Examples

Creating HTML Sections

The following example defines a section group called htmgroup of type HTML_SECTION_GROUP. It then creates a zone section in htmgroup called headline identified by the <H1> tag:

```
begin
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
ctx_ddl.add_zone_section('htmgroup', 'heading', 'H1');
end;
```

After indexing with section group htmgroup, query within the heading section by issuing a query as follows:

```
'Oracle WITHIN heading'
```

Creating Sections for <META NAME> Tags

You can create zone sections for HTML META tags when you use the HTML_SECTION_GROUP.

Consider an HTML document that has a META tag as follows:

```
<META NAME="author" CONTENT="ken">
```

To create a zone section that indexes all CONTENT attributes for the META tag whose NAME value is author:

```
begin
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
ctx_ddl.add_zone_section('htmgroup', 'author', 'meta@author');
end
```

After indexing with section group htmgroup, query the document as follows:

```
'ken WITHIN author'
```

Creating Document Type Sensitive Sections (XML Documents Only)

You have an XML document set that contains the <book> tag declared for different document types (DTDs). You want to create a distinct book section for each document type.

Assume that myDTDname is declared as an XML document type as follows:

```
<!DOCTYPE myDTDname>
<myDTDname>
 ...
```

(Note: the DOCTYPE must match the top-level tag.)

Within myDTDname, the element <book> is declared. For this tag, create a section named mybooksec that is sensitive to the tag's document type as follows:

```
begin
ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_zone_section('myxmlgroup', 'mybooksec', '(myDTDname)book');
end;
```

Notes

Repeated Sections

Zone sections can repeat. Each occurrence is treated as a separate section. For example, if <H1> denotes a heading section, they can repeat in the same documents as follows:

<H1> The Brown Fox </H1>

<H1> The Gray Wolf </H1>

Assuming that these zone sections are named `Heading`, the query *Brown WITHIN Heading* returns this document. However, a query of *(Brown and Gray) WITHIN Heading* does not.

Overlapping Sections

Zone sections can overlap each other. For example, if `<B>` and `<I>` denote two different zone sections, they can overlap in document as follows:

```
plain <B> bold <I> bold and italic </B> only italic </I>  plain
```

Nested Sections

Zone sections can nest, including themselves as follows:

```
<TD> <TABLE><TD>nested cell</TD></TABLE></TD>
```

Using the `WITHIN` operator, you can write queries to search for text in sections within sections. For example, assume the `BOOK1`, `BOOK2`, and `AUTHOR` zone sections occur as follows in documents doc1 and doc2:

doc1:

```
<book1> <author>Scott Tiger</author> This is a cool book to read.</book1>
```

doc2:

```
<book2> <author>Scott Tiger</author> This is a great book to read.</book2>
```

Consider the nested query:

```
'(Scott within author) within book1'
```

This query returns only doc1.

Related Topics

"WITHIN" on page 3-60

"Section Group Types" on page 2-66

"CREATE_SECTION_GROUP" on page 7-38

"ADD_FIELD_SECTION" on page 7-4

"ADD_SPECIAL_SECTION" on page 7-18

"REMOVE_SECTION" on page 7-67

"DROP_SECTION_GROUP" on page 7-48

COPY_POLICY

Creates a new policy from an existing policy or index.

Syntax

```
ctx_ddl.copy_policy(
    source_policy       VARCHAR2,
    policy_name         VARCHAR2
    );
```

source_policy
The name of the policy or index being copied.

policy_name
The name of the new policy copy.

The preference values are copied from the `source_policy`. Both the source policy or index and the new policy must be owned by the same database user.

CREATE_INDEX_SET

Creates an index set for CTXCAT index types. Name this index set in the parameter clause of CREATE INDEX when you create a CTXCAT index.

Syntax

```
CTX_DDL.CREATE_INDEX_SET(set_name in varchar2);
```

set_name
Specify the name of the index set. Name this index set in the parameter clause of CREATE INDEX when you create a CTXCAT index.

CREATE_POLICY

Creates a policy to use with the CTX_DOC.POLICY_* procedures and the ORA:CONTAINS function. ORA:CONTAINS is a function you use within an XPATH query expression with existsNode().

See Also: *Oracle XML DB Developer's Guide*

Syntax

```
CTX_DDL.CREATE_POLICY(
         policy_name     IN VARCHAR2,
         filter          IN VARCHAR2 DEFAULT NULL,
         section_group   IN VARCHAR2 DEFAULT NULL,
         lexer           IN VARCHAR2 DEFAULT NULL,
         stoplist        IN VARCHAR2 DEFAULT NULL,
         wordlist        IN VARCHAR2 DEFAULT NULL);
```

policy_name
Specify the name for the new policy. Policy names and Text indexes share the same namespace.

filter
Specify the filter preference to use.

section_group
Specify the section group to use. You can specify any section group that is supported by CONTEXT index.

lexer
Specify the lexer preference to use. Your INDEX_THEMES attribute must be disabled.

stoplist
Specify the stoplist to use.

wordlist
Specify the wordlist to use.

Example

Create mylex lexer preference named mylex.

```
begin
   ctx_ddl.create_preference('mylex', 'BASIC_LEXER');
   ctx_ddl.set_attribute('mylex', 'printjoins', '_-');
   ctx_ddl.set_attribute ( 'mylex', 'index_themes', 'NO');
   ctx_ddl.set_attribute ( 'mylex', 'index_text', 'YES');
end;
```

Create a stoplist preference named mystop.

```
begin
  ctx_ddl.create_stoplist('mystop', 'BASIC_STOPLIST');
  ctx_ddl.add_stopword('mystop', 'because');
  ctx_ddl.add_stopword('mystop', 'nonetheless');
  ctx_ddl.add_stopword('mystop', 'therefore');
end;
```

Create a wordlist preference named 'mywordlist'.

```
begin
 ctx_ddl.create_preference('mywordlist', 'BASIC_WORDLIST');
 ctx_ddl.set_attribute('mywordlist','FUZZY_MATCH','ENGLISH');
 ctx_ddl.set_attribute('mywordlist','FUZZY_SCORE','0');
 ctx_ddl.set_attribute('mywordlist','FUZZY_NUMRESULTS','5000');
 ctx_ddl.set_attribute('mywordlist','SUBSTRING_INDEX','TRUE');
 ctx_ddl.set_attribute('mywordlist','STEMMER','ENGLISH');
end;

exec ctx_ddl.create_policy('my_policy', NULL, NULL, 'mylex', 'mystop',
'mywordlist');
```

or

```
exec ctx_ddl.create_policy(policy_name => 'my_policy',
                           lexer => 'mylex',
                           stoplist => 'mystop',
                           wordlist => 'mywordlist');
```

Then enter the following `existsNode()` query with your own defined policy:

```
select id from xmltab
  where existsNode(doc, '/book/chapter[ ora:contains(summary,"dog or cat",
  "my_policy") >0 ]', 'xmlns:ora="http://xmlns.oracle.com/xdb" ')=1;
```

Update the policy with the following:

```
exec ctx_ddl.update_policy(policy_name => 'my_policy', lexer => 'my_new_lex');
```

Drop the policy with the following:

```
exec ctx_ddl.drop_policy(policy_name => 'my_policy');
```

CREATE_PREFERENCE

Creates a preference in the Text data dictionary. Specify preferences in the parameter string of CREATE INDEX or ALTER INDEX.

> **Caution:** CTX_DDL.CREATE_PREFERENCE does not respect the current schema as set by ALTER SESSION SET current_schema. Therefore, if you need to create or delete a preference owned by another user, then you must explicitly state this, and you must have the CREATE ANY TABLE system privilege.

Syntax

```
CTX_DDL.CREATE_PREFERENCE(preference_name  in varchar2,
                          object_name      in varchar2);
```

preference_name
Specify the name of the preference to be created.

object_name
Specify the name of the preference type.

> **See Also:** For a complete list of preference types and their associated attributes, see Chapter 2, "Oracle Text Indexing Elements"

Examples

Creating Text-only Index

The following example creates a lexer preference that specifies a text-only index. It does so by creating a BASIC_LEXER preference called my_lexer with CTX_DDL.CREATE_PREFERENCE. It then calls CTX_DDL.SET_ATTRIBUTE twice, first specifying YES for the INDEX_TEXT attribute, then specifying NO for the INDEX_THEMES attribute.

```
begin
ctx_ddl.create_preference('my_lexer', 'BASIC_LEXER');
ctx_ddl.set_attribute('my_lexer', 'INDEX_TEXT', 'YES');
ctx_ddl.set_attribute('my_lexer', 'INDEX_THEMES', 'NO');
end;
```

Specifying File Data Storage

The following example creates a data storage preference called mypref that tells the system that the files to be indexed are stored in the operating system. The example then uses CTX_DDL.SET_ATTRIBUTE to set the PATH attribute of to the directory /docs.

```
begin
ctx_ddl.create_preference('mypref', 'FILE_DATASTORE');
ctx_ddl.set_attribute('mypref', 'PATH', '/docs');
end;
```

> **See Also:** For more information about data storage, see "Datastore Types" on page 2-2

Creating Master/Detail Relationship

Use CTX_DDL.CREATE_PREFERENCE to create a preference with DETAIL_DATASTORE. Use CTX_DDL.SET_ATTRIBUTE to set the attributes for this preference. The following example shows how this is done:

```
begin
ctx_ddl.create_preference('my_detail_pref', 'DETAIL_DATASTORE');
ctx_ddl.set_attribute('my_detail_pref', 'binary', 'true');
ctx_ddl.set_attribute('my_detail_pref', 'detail_table', 'my_detail');
ctx_ddl.set_attribute('my_detail_pref', 'detail_key', 'article_id');
ctx_ddl.set_attribute('my_detail_pref', 'detail_lineno', 'seq');
ctx_ddl.set_attribute('my_detail_pref', 'detail_text', 'text');
end;
```

See Also: For more information about master/detail, see "DETAIL_DATASTORE" on page 2-6

Specifying Storage Attributes

The following examples specify that the index tables are to be created in the foo tablespace with an initial extent of 1K:

```
begin
ctx_ddl.create_preference('mystore', 'BASIC_STORAGE');
ctx_ddl.set_attribute('mystore', 'I_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'K_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'R_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'S_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'N_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'I_INDEX_CLAUSE',
                      'tablespace foo storage (initial 1K)');
end;
```

See Also: Storage Types on page 2-64

Creating Preferences with No Attributes

When you create preferences with types that have no attributes, you need only create the preference, as in the following example which sets the filter to the NULL_FILTER:

```
begin
ctx_ddl.create_preference('my_null_filter', 'NULL_FILTER');
end;
```

Notes

If s_table_clause is specified for a storage preference in an index without SDATA, then it has no effect on the index, and the index creation will still succeed.

Related Topics

"SET_ATTRIBUTE" on page 7-73

"DROP_PREFERENCE" on page 7-47

"CREATE INDEX" on page 1-36

"ALTER INDEX" on page 1-2

Chapter 2, "Oracle Text Indexing Elements"

CREATE_SECTION_GROUP

Creates a section group for defining sections in a text column.

When you create a section group, you can add to it zone, field, or special sections with ADD_ZONE_SECTION, ADD_FIELD_SECTION, ADD_MDATA_SECTION, or ADD_SPECIAL_SECTION.

When you index, name the section group in the parameter string of CREATE INDEX or ALTER INDEX.

After indexing, query within your defined sections with the WITHIN operator.

Syntax

```
CTX_DDL.CREATE_SECTION_GROUP(
  group_name     in    varchar2,
  group_type     in    varchar2
);
```

group_name
Specify the section group name to create as `[user.]section_group_name`. This parameter must be unique within an owner.

group_type
Specify section group type. The group_type parameter can be one of:

Section Group Preference	Description
`NULL_SECTION_GROUP`	Use this group type when you define no sections or when you define *only* SENTENCE or PARAGRAPH sections. This is the default.
`BASIC_SECTION_GROUP`	Use this group type for defining sections where the start and end tags are of the form `<A>` and `</A>`. Note: This group type dopes not support input such as unbalanced parentheses, comments tags, and attributes. Use `HTML_SECTION_GROUP` for this type of input.
`HTML_SECTION_GROUP`	Use this group type for indexing HTML documents and for defining sections in HTML documents.
`XML_SECTION_GROUP`	Use this group type for indexing XML documents and for defining sections in XML documents.

Section Group Preference	Description
AUTO_SECTION_GROUP	Use this group type to automatically create a zone section for each start-tag/end-tag pair in an XML document. The section names derived from XML tags are case sensitive as in XML. Attribute sections are created automatically for XML tags that have attributes. Attribute sections are named in the form attribute@tag. Stop sections, empty tags, processing instructions, and comments are not indexed. The following limitations apply to automatic section groups: You cannot add zone, field, or special sections to an automatic section group. Automatic sectioning does not index XML document types (root elements.) However, you can define stop sections with document type. The length of the indexed tags, including prefix and namespace, cannot exceed 64 bytes. Tags longer than this are not indexed.
PATH_SECTION_GROUP	Use this group type to index XML documents. Behaves like the AUTO_SECTION_GROUP. The difference is that with this section group you can do path searching with the INPATH and HASPATH operators. Queries are also case-sensitive for tag and attribute names.
NEWS_SECTION_GROUP	Use this group for defining sections in newsgroup formatted documents according to RFC 1036.

Example

The following command creates a section group called htmgroup with the HTML group type.

```
begin
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
end;
```

The following command creates a section group called auto with the AUTO_SECTION_GROUP group type to be used to automatically index tags in XML documents.

```
begin
ctx_ddl.create_section_group('auto', 'AUTO_SECTION_GROUP');
end;
```

Related Topics

"WITHIN" on page 3-60

"Section Group Types" on page 2-66

"ADD_ZONE_SECTION" on page 7-28

"ADD_FIELD_SECTION" on page 7-4

"ADD_MDATA_SECTION" on page 7-12

"ADD_SPECIAL_SECTION" on page 7-18

"REMOVE_SECTION" on page 7-67

"DROP_SECTION_GROUP" on page 7-48

CREATE_SHADOW_INDEX

Creates index metadata (or policy) for the specified index. If the index is not partitioned, then it also creates the index tables. This procedure is only supported in Enterprise Edition of Oracle Database.

The following changes are not supported:

- Transition from non-composite domain index to composite, or changing the composite domain index columns.
- Rebuild indexes that have partitioned index tables, for example, $I, $P, $K.

> **Note:** For a partitioned index, you must first call this procedure to create the shadow index metadata. This procedure will not create index tables. It has no effect on query, DML, sync, or optimize operations.

Syntax

```
CTX_DDL.CREATE_SHADOW_INDEX(
  idx_name          IN VARCHAR2,
  parameter_string  IN VARCHAR2 DEFAULT NULL,
  parallel_degree   IN NUMBER, DEFAULT 1
);
```

idx_name
The name of a valid CONTEXT indextype.

parameter_string
For non-partitioned index, the same string as in ALTER INDEX. For partitioned index, the same string as in ALTER INDEX PARAMETER.

parallel_degree
Reserved for future use. Specify the degree of parallelism. Parallel operation is not currently supported.

Example

Example 7–1 Scheduled Global Index Recreate (Incremental Rebuild)

In this example, you have the finest control over each stage of RECREATE_INDEX_ONLINE. Since SYNC_INDEX can take a time limit, you can limit SYNC_INDEX during non-business hours and incrementally recreate the index.

```
/* create lexer and original index */
exec ctx_ddl.create_preference('us_lexer','basic_lexer');
create index idx on tbl(text) indextype is ctxsys.context
  parameters('lexer us_lexer');

/* create a new lexer */
begin
  ctx_ddl.create_preference('e_lexer','basic_lexer');
  ctx_ddl.set_attribute('e_lexer','base_letter','yes');
  ctx_ddl.create_preference('m_lexer','multi_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','default','us_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','e','e_lexer');
```

```
end;
/

/* add new language column to the table for multi-lexer */
alter table tbl add(lang varchar2(10) default 'us');

/* create shadow index */
exec ctx_ddl.create_shadow_index('idx',
  'replace lexer m_lexer language column lang NOPOPULATE');

declare
  idxid integer;
begin
  /* figure out shadow index name */
  select idx_id into idxid from ctx_user_indexes
     where idx_name ='IDX';
  /* populate pending */
  ctx_ddl.populate_pending('RIO$'||idxid);
  /* time limited sync */
  ctx_ddl.sync_index(idx_name =>'RIO$'||idxid,
                     maxtime =>480);
  /* more sync until no pending rows for the shadow index */
end;
/* swap in the shadow index */
exec ctx_ddl.exchange_shadow_index('idx');
```

Notes

The index name for the shadow index is `RIO$index_id`. By default it will also populate index tables for non-partitioned indexes, unless `NOPOPULATE` is specified in `CREATE INDEX` or in `ALTER INDEX`. For a local partitioned index, it will only create index metadata without creating the index tables for each partition. Each index can have only one shadow index.

When building a non-partitioned index online, you can first call this procedure to create index metadata and index tables. If you specify `POPULATE`, then this procedure will populate the index, but will not do swapping. You can schedule the swapping at a later, preferred time.

If you specify `NOPOPULATE`, it will only create metadata for the index tables, but will not populate them. You must perform `POPULATE_PENDING` (CTX_DDL.POPULATE_PENDING) to populate the pending queues after running this procedure, and then sync the indexes. This is referred to as *incremental recreate*.

Queries are all processed normally when this procedure is running.

If `POPULATE` is specified, then DML is blocked for a very short time at the beginning of populate, after which all further DML is logged into an online pending queue and processed later.

Sync with CTX_DDL.SYNC_INDEX runs normally on the index. OPTIMIZE_INDEX runs without doing anything, but does not return an error.

Related Topics

See also `POPULATE | NOPOPULATE` in ALTER INDEX and CREATE INDEX in Chapter 1, "Oracle Text SQL Statements and Operators", and CTX_DDL.DROP_SHADOW_INDEX, CTX_DDL.EXCHANGE_SHADOW_INDEX, CTX_DDL.SYNC_INDEX, and CTX_DDL.POPULATE_PENDING in this chapter.

CREATE_STOPLIST

Use this procedure to create a new, empty stoplist. Stoplists can contain words or themes that are not to be indexed.

You can also create multi-language stoplists to hold language-specific stopwords. A multi-language stoplist is useful when you index a table that contains documents in different languages, such as English, German, and Japanese. When you do so, the text table must contain a language column.

Add either stopwords, stopclasses, or stopthemes to a stoplist using ADD_STOPWORD, ADD_STOPCLASS, or ADD_STOPTHEME. Specify a stoplist in the parameter string of CREATE INDEX or ALTER INDEX to override the default stoplist CTXSYS.DEFAULT_STOPLIST.

Syntax

```
CTX_DDL.CREATE_STOPLIST(
stoplist_name IN VARCHAR2,
stoplist_type IN VARCHAR2 DEFAULT 'BASIC_STOPLIST');
```

stoplist_name
Specify the name of the stoplist to be created.

stoplist_type
Specify `BASIC_STOPLIST` to create a stoplist for a single language. This is the default.

Specify `MULTI_STOPLIST` to create a stoplist with language-specific stopwords.

At indexing time, the language column of each document is examined, and only the stopwords for that language are eliminated. At query time, the session language setting determines the active stopwords, like it determines the active lexer when using the multi-lexer.

> **Note:** When indexing a multi-language table with a multi-language stoplist, the table must have a language column.

Examples

Example 7–2 Single Language Stoplist

The following example creates a stoplist called `mystop`:

```
begin
ctx_ddl.create_stoplist('mystop', 'BASIC_STOPLIST');
end;
```

Example 7–3 Multi-Language Stoplist

The following example creates a multi-language stoplist called `multistop` and then adds tow language-specific stopwords:

```
begin
ctx_ddl.create_stoplist('multistop', 'MULTI_STOPLIST');
ctx_ddl.add_stopword('mystop', 'Die','german');
ctx_ddl.add_stopword('mystop', 'Or','english');
end;
```

Related Topics

"ADD_STOPWORD" on page 7-24

"ADD_STOPCLASS" on page 7-20

"ADD_STOPTHEME" on page 7-23

"DROP_STOPLIST" on page 7-50

"CREATE INDEX" on page 1-36

"ALTER INDEX" on page 1-2

Appendix E, "Oracle Text Supplied Stoplists"

DROP_INDEX_SET

Drops a CTXCAT index set created with CTX_DDL.CREATE_INDEX_SET.

Syntax

```
CTX_DDL.DROP_INDEX_SET(
     set_name    IN VARCHAR2
);
```

set_name
Specify the name of the index set to drop.

Dropping an index set drops all of the sub-indexes it contains.

DROP_POLICY

Drops a policy created with CTX_DDL.CREATE_POLICY.

Syntax

```
CTX_DDL.DROP_POLICY(
     policy_name    IN VARCHAR2
);
```

policy_name
Specify the name of the policy to drop.

DROP_PREFERENCE

The DROP_PREFERENCE procedure deletes the specified preference from the Text data dictionary. Dropping a preference does not affect indexes that have already been created using that preference.

Syntax

```
CTX_DDL.DROP_PREFERENCE(
     preference_name    IN VARCHAR2
);
```

preference_name
Specify the name of the preference to be dropped.

Example

The following example drops the preference my_lexer.

```
begin
ctx_ddl.drop_preference('my_lexer');
end;
```

Related Topics

See also CTX_DDL.CREATE_PREFERENCE.

DROP_SECTION_GROUP

The DROP_SECTION_GROUP procedure deletes the specified section group, as well as all the sections in the group, from the Text data dictionary.

Syntax

```
CTX_DDL.DROP_SECTION_GROUP(
     group_name      IN VARCHAR2
);
```

group_name
Specify the name of the section group to delete.

Example

The following example drops the section group htmgroup and all its sections:

```
begin
ctx_ddl.drop_section_group('htmgroup');
end;
```

Related Topics

See also CTX_DDL.CREATE_SECTION_GROUP.

DROP_SHADOW_INDEX

Drops a shadow index for the specified index. When you drop a shadow index, if it is partitioned, then its metadata and the metadata of all this shadow index's partitions are dropped. This procedure also drops all the shadow index tables and cleans up any online pending queue.

Syntax

```
CTX_DDL.DROP_SHADOW_INDEX(
     idx_name       in VARCHAR2
);
```

idx_name
The name of a valid CONTEXT indextype.

Example

The following example drops the shadow index myshadowidx:

```
begin
ctx_ddl.drop_shadow_index('myshadowidx');
end;
```

Related Topics

See also CTX_DDL.CREATE_SHADOW_INDEX.

DROP_STOPLIST

Drops a stoplist from the Text data dictionary. When you drop a stoplist, you must re-create or rebuild the index for the change to take effect.

Syntax

```
CTX_DDL.DROP_STOPLIST(stoplist_name in varchar2);
```

stoplist_name
Specify the name of the stoplist.

Example

The following example drops the stoplist `mystop`:

```
begin
ctx_ddl.drop_stoplist('mystop');
end;
```

Related Topics

See also CTX_DDL.CREATE_STOPLIST.

EXCHANGE_SHADOW_INDEX

This procedure swaps the index (or index partition) metadata and index (or index partition) data.

For non-partitioned indexes, this procedure swaps both the metadata and the index data, and processes the online pending queue.

Syntax

```
CTX_DDL.EXCHANGE_SHADOW_INDEX(
     idx_name        IN VARCHAR2
     partition_name  IN VARCHAR2 default NULL
);
```

idx_name
Specify the name of the CONTEXT indextype.

partition_name
Specify the name of the shadow index partition. May also be NULL.

Example

Example 7–4 Global Index Recreate with Scheduled Swap

This example demonstrates running CTX_DDL.EXCHANGE_SHADOW_INDEX during non-business hours when query failures and DML blocking can be tolerated.

```
/* create lexer and original index */
exec ctx_ddl.create_preference('us_lexer','basic_lexer');
create index idx on tbl(text) indextype is ctxsys.context
  parameters('lexer us_lexer');

/* create a new lexer */
begin
  ctx_ddl.create_preference('e_lexer','basic_lexer');
  ctx_ddl.set_attribute('e_lexer','base_letter','yes');
  ctx_ddl.create_preference('m_lexer','multi_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','default','us_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','e','e_lexer');
end;
/

/* add new language column to the table for multi-lexer */
alter table tbl add(lang varchar2(10) default 'us');

/* recreate index online with the new multip-lexer */
exec ctx_ddl.create_shadow_index('idx',
  'replace lexer m_lexer language column lang');
exec ctx_ddl.exchange_shadow_index('idx');
```

Notes

Using EXCHANGE_SHADOW_INDEX with Non-partitioned Indexes

For non-partitioned indexes, this procedure will swap both metadata and index data, and will process the online pending queue.

Queries will return *column not indexed* errors when swapping metadata and index data, but queries are processed normally when processing online pending queue. The period of errors being raised should be short.

If you specify `POPULATE` when you create the shadow index, and if many DML operations have been issued since the creation of the shadow index, then there could be a large pending queue. However, if you use *incremental recreate*, that is, specify `NOPOPULATE` when you create the shadow index, and you then populate the pending queue and sync, then the online pending queue is always empty no matter how many DML operations have occurred since `CREATE_SHADOW_INDEX` was issued.

When this procedure is running, DML will first fail with an error about index being in in-progress status. After that DML could be blocked (hang) if there are rows in online pending queue that need to be reapplied.

Note: When this procedure is running, DML statements will fail with an error that the index is in "in-progress status." If, when this error occurs, there are rows in the online pending queue that need to be reapplied, then the DML could be blocked and hang.

Using EXCHANGE_SHADOW_INDEX with Partitioned Indexes

For partitions that are recreated with `NOSWAP`: when the index is partitioned, and if *partition_name* is a valid index partition, then this procedure will swap the index partition data and the index partition metadata, and will process the online pending queue for this partition.

This procedure swaps only one partition at a time. When you run this procedure on partitions that are recreated with `NOSWAP`:

- Queries that span multiple partitions will not return consistent results across all partitions.
- Queries on the partition that is being swapped will return errors.
- Queries on partitions that are already swapped will be based on the new index.
- Queries on the partitions that haven't been swapped will be based on the old index.

If the *partition_name* is NULL, then this procedure will swap the index metadata. Run this procedure as the last step when recreating a local partitioned index online.

Related Topics

See also CTX_DDL.RECREATE_INDEX_ONLINE, CTX_DDL.CREATE_SHADOW_INDEX, and CTX_DDL.DROP_SHADOW_INDEX.

OPTIMIZE_INDEX

Use this procedure to optimize the index. Optimize your index after you synchronize it. Optimizing an index removes old data and minimizes index fragmentation, which can improve query response time. Querying and DML may proceed while optimization takes place.

You can optimize in fast, full, rebuild, token, or token-type mode.

- Fast mode compacts data but does not remove rows.
- Full mode compacts data and removes rows.
- Optimize in rebuild mode rebuilds the `$I` table (the inverted list table) in its entirety. Rebuilding an index is often significantly faster than performing a full optimization, and is more likely to result in smaller indexes, especially if the index is heavily fragmented.

  Rebuild optimization creates a more compact copy of the `$I` table, and then switches the original `$I` table and the copy. The rebuild operation will therefore require enough space to store the copy as well as the original. (If redo logging is enabled, then additional space is required in the redo log as well.) At the end of the rebuild operation, the original `$I` table is dropped, and the space can be reused.

  Optimize in rebuild mode supports partitioning on the `$I` table via the `i_table_clause` attribute of the `basic_storage` preference with the following limitations:

  - The `i_index_clause` must specify using a local btree index if the `$I` table is partitioned.
  - Partitioning schemes on the `token_first`, `token_last`, or `token_count` columns are not allowed.
- In token mode, specify a specific token to be optimized (for example, all rows with documents containing the word *elections*). Use this mode to optimize index tokens that are frequently searched, without spending time on optimizing tokens that are rarely referenced. An optimized token can improve query response time (but only for queries on that token).
- Token-type optimization is similar to token mode, except that the optimization is performed on field sections or `MDATA` sections (for example, sections with an `<A>` tag). This is useful in keeping critical field or `MDATA` sections optimal.

A common strategy for optimizing indexes is to perform regular token optimizations on frequently referenced terms, and to perform rebuild optimizations less frequently. (Use CTX_REPORT.QUERY_LOG_SUMMARY to find out which queries are made most frequently.) You can perform full, fast, or token-type optimizations instead of token optimizations.

Some users choose to perform frequent time-limited full optimizations along with occasional rebuild optimizations.

Note: Optimizing an index can result in better response time only if you insert, delete, or update documents in the base table after your initial indexing operation.

Using this procedure to optimize the index is recommended over using the ALTER INDEX statement.

Optimization of a large index may take a long time. To monitor the progress of a lengthy optimization, log the optimization with CTX_OUTPUT.START_LOG and check the resultant logfile from time to time.

Note that, unlike serial optimize full, CTX_DDL.OPTIMIZE_INDEX() run with optlevel of FULL and parallel_degree > 1 is not resumable. That is, it will not resume from where it left after a time-out or failure.

Note: There is a very small window of time when a query might fail in CTX_DDL.OPTIMIZE_INDEX REBUILD mode when the $I table is being swapped with the optimized shadow $I table.

Syntax

```
CTX_DDL.OPTIMIZE_INDEX(
idx_name        IN  VARCHAR2,
optlevel        IN  VARCHAR2,
maxtime         IN  NUMBER DEFAULT NULL,
token           IN VARCHAR2 DEFAULT NULL,
part_name       IN VARCHAR2 DEFAULT NULL,
token_type      IN NUMBER DEFAULT NULL,
parallel_degree IN NUMBER DEFAULT 1);
);
```

idx_name

Specify the name of the index. If you do not specify an index name, then Oracle Text chooses a single index to optimize.

optlevel

Specify optimization level as a string. You can specify one of the following methods for optimization:

optlevel value	Description
FAST or CTX_DDL.OPTLEVEL_FAST	This method compacts fragmented rows. However, old data is not removed. FAST optimization is not supported for CTXCAT indexes. FAST optimization will not optimize $S index table.
FULL or CTX_DDL.OPTLEVEL_FULL	In this mode you can optimize the entire index or a portion of the index. This method compacts rows and removes old data (deleted rows). Optimizing in full mode runs even when there are no deleted rows. Full optimization is not supported for CTXCAT indexes.
REBUILD or CTX_DDL.OPTLEVEL_REBUILD	This optlevel rebuilds the $I table (the inverted list table) to produce more compact token info rows. Like FULL optimize, this mode also deletes information pertaining to deleted rows of the base table. REBUILD is not supported for CTCAT, CTXRULE, or CTXXPATH indexes. REBUILD is not supported when the $I table is partitioned.

optlevel value	Description
TOKEN or CTX_DDL.OPTLEVEL_TOKEN	This method lets you specify a specific token to be optimized. Oracle Text does a full optimization on the token you specify with token. If no token type is provided, 0 (zero) will be used as the default. Use this method to optimize those tokens that are searched frequently. Token optimization is not supported for CTXCAT, CTXRULE, and CTXXPATH indexes.
TOKEN_TYPE or CTX_DDL.OPTLEVEL_TOKEN_TYPE	This optlevel optimizes on demand all tokens in the index matching the input token type. When optlevel is TOKEN_TYPE, *token_type* must be provided.TOKEN_TYPE performs FULL optimize on any token of the input *token_type*. Like a TOKEN optimize, TOKEN_TYPE optimize does not change the FULL optimize state, and runs to completion on each invocation. Token_type optimization is not supported for CTXCAT, CTXRULE, and CTXXPATH indexes.

The behavior of CTX_DDL.OPTIMIZE_INDEX with respect to the $S index table is as follows:

optlevel value	Will Optimize $S Index Table Yes/No	Notes
FAST or CTX_DDL.OPTLEVEL_FAST	No	
FULL or CTX_DDL.OPTLEVEL_FULL	Yes	The optimize process will optimize $I table first. Once $I table optimize is finished, CTX_DDL.OPTIMIZE_INDEX will continue on to optimize $S index table. MAXTIME will also be honored. Once CTX_DDL.OPTIMIZE_INDEX completes optimizing $S rows for a given SDATA_ID, it will check MAXTIME and exit if total elapsed time (including time taken to optimize $I) exceeds specified MAXTIME. The next CTX_DDL.OPTIMIZE_INDEX with optlevel=>'FULL' will pick up where it left off. $S table optimize will be done in serial.
REBUILD or CTX_DDL.OPTLEVEL_REBUILD	Yes	$S optimize will start after $I rebuild finishes. $S optimize in this case will be processed the same way as $S optimize in FULL mode. $S table is optimized in place, not rebuilt. **Note:** If for some reason $S optimize exits abnormally, then it is recommended that you use optlevel=>TOKEN_TYPE to optimize $S to avoid rebuilding the $I table again. $S table optimize will be done in serial.
TOKEN or CTX_DDL.OPTLEVEL_TOKEN	No	

optlevel value	Will Optimize $S Index Table Yes/No	Notes
TOKEN_TYPE or CTX_DDL.OPTLEVEL_TOKEN_TYPE	Yes	You can optimize $S rows for a given SDATA_ID by setting optlevel => TOKEN_TYPE and the TOKEN_TYPE parameter to the target SDATA_ID.

maxtime
Specify maximum optimization time, in minutes, for FULL optimize.

When you specify the symbol CTX_DDL.MAXTIME_UNLIMITED (or pass in NULL), the entire index is optimized. This is the default.

token
Specify the token to be optimized.

part_name
If your index is a local index, then you must specify the name of the index partition to synchronize otherwise an error is returned.

If your index is a global, non-partitioned index, then specify NULL, which is the default.

token_type
Specify the token_type to be optimized.

parallel_degree
Specify the parallel degree as a number for parallel optimization. The actual parallel degree depends on your resources. Note that when using REBUILD, setting parallel_degree to a value greater than 1 still results in serial execution.

Because the following optlevel values are executed serially, this setting is ignored for them:

- TOKEN or CTX_DDL.OPTLEVEL_TOKEN
- FAST or CTX_DDL.OPTLEVEL_FAST

Examples

The following two examples are equivalent ways of optimizing an index using fast optimization:

```
begin
  ctx_ddl.optimize_index('myidx','FAST');
end;

begin
  ctx_ddl.optimize_index('myidx',CTX_DDL.OPTLEVEL_FAST);
end;
```

The following example optimizes the index token *Oracle*:

```
begin
  ctx_ddl.optimize_index('myidx','token', TOKEN=>'Oracle');
end;
```

To optimize all tokens of field section MYSEC in index MYINDEX:

```
begin
```

```
  ctx_ddl.optimize_index('myindex', ctx_ddl.optlevel_token_type,
     token_type=> ctx_report.token_type('myindex','field mysec text'));
end;
```

Notes

You can run CTX_DDL.SYNC_INDEX and CTX_DDL.OPTIMIZE_INDEX at the same time. You can also run CTX_DDL.SYNC_INDEX and CTX_DDL.OPTIMIZE_INDEX with parallelism at the same time. However, you should not:

- Run CTX_DDL.SYNC_INDEX with parallelism at the same time as CTX_DDL.OPTIMIZE_INDEX
- Run CTX_DDL.SYNC_INDEX with parallelism at the same time as CTX_DDL.OPTIMIZE_INDEX with parallelism.

If you should run one of these combinations, no error is generated; however, one operation will wait until the other is done.

Related Topics

See also CTX_DDL.SYNC_INDEX and ALTER INDEX.

POPULATE_PENDING

This procedure populates the pending queue with every rowid in the base table or table partition. This procedure is only supported for CONTEXT indexes.

This procedure is valuable for large installations that cannot afford to have the indexing process run continuously, and, therefore, need finer control over creating text indexes. The preferred method is to create an empty index, place all the rowids into the pending queue, and build the index through CTX_DDL.SYNC_INDEX.

Syntax

```
ctx_ddl.populate_pending(
  idx_name  IN VARCHAR2,
  part_name IN VARCHAR2 DEFAULT NULL
);
```

idx_name
Name of the CONTEXT indextype.

part_name
Name of the index partition, if any. Must be provided for local partitioned indexes and must be NULL for global, non-partitioned indexes.

Notes

The SYNC_INDEX is blocked for the duration of the processing. The index unit must be totally empty (idx_docid_count = 0, idx_nextid = 1). The rowids of rows waiting to be indexed are inserted into table ctxsys.dr$pending. You should ensure that there is sufficient space in this table to hold the rowids of the base table.

Related Topics

See also SYNC_INDEX, CREATE_SHADOW_INDEX, DROP_SHADOW_INDEX, EXCHANGE_SHADOW_INDEX, RECREATE_INDEX_ONLINE.

RECREATE_INDEX_ONLINE

Recreates the specified index, or recreates the passed-in index partition if the index is local partitioned. For global non-partitioned indexes, this is a one-step procedure. For local partitioned indexes, this procedure must be run separately on every partition after first using CREATE_SHADOW_INDEX to create a shadow policy (or metadata). This procedure is only supported in Enterprise Edition of Oracle Database.

The following changes are not supported:

- Transitioning from non-composite domain index to composite, or changing the composite domain index columns.
- Rebuilding indexes that have partitioned index tables, for example, `$I`, `$P`, `$K`.

Syntax

```
CTX_DDL.RECREATE_INDEX_ONLINE(
   idx_name          IN VARCHAR2,
   parameter_string  IN VARCHAR2 default NULL,
   parallel_degree   IN NUMBER default 1,
   partition_name    IN VARCHAR2 default NULL
);
```

idx_name
The name of a valid `CONTEXT` indextype.

parameter_string
If the index is a global non-partitioned index, specify the same index-level parameter string as in `ALTER INDEX`. Must start with `REPLACE`, if it is not `NULL`. Optionally specify `SWAP` or `NOSWAP`. The default is `SWAP`.

parallel_degree
Reserved for future use. Specify the degree of parallelism. Parallel operation is not supported in the current release.

partition_name
Specify the name of a valid index partition for a local partitioned index. Otherwise, the default is `NULL`. If the index is partitioned, then first pass a partition name, and then specify the partition-level parameter string for `ALTER INDEX REBUILD PARTITION`.

Examples

Example 7–5 Recreate Simple Global Index

The following example creates an index `idx` with a `BASIC_LEXER`-based preference `us_lexer`. It then recreates the index with a new `MULTI_LEXER` based preference `m_lexer` in one step. You can use this one step approach when you do not mind that a query might fail for a small window of time at the end of the operation, and DML might get blocked at the beginning for a short time and again at the end.

```
/* create lexer and original index */
exec ctx_ddl.create_preference('us_lexer','basic_lexer');
create index idx on tbl(text) indextype is ctxsys.context
  parameters('lexer us_lexer');

/* create a new lexer */
begin
```

```
  ctx_ddl.create_preference('e_lexer','basic_lexer');
  ctx_ddl.set_attribute('e_lexer','base_letter','yes');
  ctx_ddl.create_preference('m_lexer','multi_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','default','us_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','e','e_lexer');
end;
/

/* add new language column to the table for multi-lexer */
alter table tbl add(lang varchar2(10) default 'us');

/* recreate index online with the new multip-lexer */
exec ctx_ddl.recreate_index_online('idx',
  'replace lexer m_lexer language column lang');
```

Example 7–6 Local Index Recreate with All-At-Once Swap

The following example creates a local partitioned index `idxp` with a basic lexer `us_lexer`. It has two index partitions `idx_p1` and `idx_p2`. It then recreates a local partitioned index `idxp` online with partition `idx_p1`, which will have a new storage preference `new_store`. The swapping of the partition metadata and index partition data occur at the end. In this example, queries spanning multiple partitions return consistent results across partitions when recreate is in process, except at the end when EXCHANGE_SHADOW_INDEX is running. The extra space required is the combined index size of partition `idx_p1` and `idx_p2`.

```
/* create a basic lexer and a local partition index with the lexer*/
exec ctx_ddl.create_preference('us_lexer','basic_lexer');
create index idxp on tblp(text) indextype is ctxsys.context local
  (partition idx_p1,
   partition idx_p2)
 parameters('lexer us_lexer');

/* create new preferences  */
begin
  ctx_ddl.create_preference('my_store','basic_storage');
  ctx_ddl.set_attribute('my_store','i_table_clause','tablespace tbs');
end;
/
begin
  ctx_ddl.create_preference('e_lexer','basic_lexer');
  ctx_ddl.set_attribute('e_lexer','base_letter','yes');
  ctx_ddl.create_preference('m_lexer','multi_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','default','us_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','e','e_lexer');
end;
/

/* add new language column */
alter table tblp add column (lang varchar2(10) default 'us');

/* create a shadow policy with a new lexer */
exec ctx_ddl.create_shadow_index('idxp', null,
  'replace lexer m_lexer language column lang');

/* recreate every index partition online without swapping */
exec ctx_ddl.recreate_index_online('idxp',
   'replace storage my_store NOSWAP', 1, 'idx_p1');
exec ctx_ddl.recreate_index_online('idxp','replace NOSWAP',1,'idx_p2');
```

```
/* exchange in shadow index partition all at once */
exec ctx_ddl.exchange_shadow_index('idxp',
      'idx_p1') /* exchange index partition data*/
exec ctx_ddl.exchange_shadow_index('idxp',
      'idx_p2') /* exchange index partition data*/

/* exchange in shadow index metadata */
exec ctx_ddl.exchange_shadow_index('idxp')
```

Example 7–7 Local Index Recreate with Per-Partition Swap

This example performs the same tasks as Example 7–6, "Local Index Recreate with All-At-Once Swap", except that each index partition is swapped in as it is completed. Queries across all partitions may return inconsistent results in this example.

```
/* create a basic lexer and a local partition index with the lexer*/
exec ctx_ddl.create_preference('us_lexer','basic_lexer');
create index idxp on tblp(text) indextype is ctxsys.context local
  (partition idx_p1,
   partition idx_p2)
 parameters('lexer us_lexer');

/* create new preferences  */
begin
  ctx_ddl.create_preference('my_store','basic_storage');
  ctx_ddl.set_attribute('my_store','i_table_clause','tablespace tbs');
end;
/
begin
  ctx_ddl.create_preference('e_lexer','basic_lexer');
  ctx_ddl.set_attribute('e_lexer','base_letter','yes');
  ctx_ddl.create_preference('m_lexer','multi_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','default','us_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','e','e_lexer');
end;
/

/* add new language column */
alter table tblp add column (lang varchar2(10) default 'us');

/* create a shadow policy with a new lexer *
exec ctx_ddl.create_shadow_index('idxp',
  'replace lexer m_lexer language column lang');

/* recreate every index partition online and swap (default) */
exec ctx_ddl.recreate_index_online('idxp',
       'replace storage my_store', 1, 'idx_p1');
exec ctx_ddl.recreate_index_online('idxp', 'replace SWAP', 1, 'idx_p2',

    /* exchange in shadow index metadata */
exec ctx_ddl.exchange_shadow_index('idxp')
```

Example 7–8 Scheduled Local Index Recreate with All-At-Once Swap

This example shows the incremental recreation of a local partitioned index, where partitions are all swapped at the end.

```
/* create a basic lexer and a local partition index with the lexer*/
exec ctx_ddl.create_preference('us_lexer','basic_lexer');
create index idxp on tblp(text) indextype is ctxsys.context local
```

```
  (partition idx_p1,
   partition idx_p2)
 parameters('lexer us_lexer');

/* create new preferences  */
begin
  ctx_ddl.create_preference('my_store','basic_storage');
  ctx_ddl.set_attribute('my_store','i_table_clause','tablespace tbs');
end;
/
begin
  ctx_ddl.create_preference('e_lexer','basic_lexer');
  ctx_ddl.set_attribute('e_lexer','base_letter','yes');
  ctx_ddl.create_preference('m_lexer','multi_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','default','us_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','e','e_lexer');
end;
/

/* add new language column */
alter table tblp add column (lang varchar2(10) default 'us');

/* create a shadow policy with a new lexer *
exec ctx_ddl.create_shadow_index('idxp',
  'replace lexer m_lexer language column lang');
/* create shadow partition with new storage preference */
exec ctx_ddl.recreate_index_online('idxp', 'replace storage ctxsys.default_storage
nopopulate',1,'idx_p1');
exec ctx_ddl.recreate_index_online('idxp', 'replace storage ctxsys.default_storage
nopopulate',1,'idx_p2');

declare
  idxid integer;
  ixpid integer;
begin
  select idx_id into idxid from ctx_user_indexes
    where idx_name = 'IDXP';
  select ixp_id into ixpid from ctx_user_index_partitions
    where ixp_index_name = 'IDXP'
          and ixp_index_partition_name = 'IDX_P1';
  /* populate pending */
  ctx_ddl.populate_pending('RIO$'||idxid, 'RIO$'||idxid||'#'||ixpid);
  /* incremental sync
  ctx_ddl.sync_index('RIO$'||idxid, null, 'RIO$'||idxid||'#'||ixpid,
                     maxtime=>400);
  /* more incremental sync until no more pending rows */

  select ixp_id into ixpid from ctx_user_index_partitions
    where ixp_index_name = 'IDXP'
          and ixp_index_partition_name = 'IDX_P2';
  /* populate pending */
  ctx_ddl.populate_pending('RIO$'||idxid, 'RIO$'||idxid||'#'||ixpid);
  /* incremental sync
  ctx_ddl.sync_index('RIO$'||idxid, null, 'RIO$'||idxid||'#'||ixpid,
                     maxtime=>400);
  /* more incremental sync until no more pending rows */
end;
/

exec ctx_ddl.exchange_shadow_index('idxp','idx_p1');
```

```
exec ctx_ddl.exchange_shadow_index('idxp','idx_p2');
exec ctx_ddl.exchange_shadow_index('idxp');
```

Example 7–9 Schedule Local Index Recreate with Per-Partition Swap

For incremental recreate where partitions are swapped as they becomes available, follow the steps in example Example 7–8, "Scheduled Local Index Recreate with All-At-Once Swap", except instead of waiting until all syncs are finished before starting exchange shadow index, EXCHANGE_SHADOW_INDEX is done for each partition right after sync is finished.

Notes

Using RECREATE_INDEX_ONLINE with Global Non-partitioned Indexes

For global indexes, this procedure provides a one-step process to recreate an index online. It recreates an index, with new preference values, while preserving base table DML and query capability during the recreate process.

> **Note:** Because the new index is created alongside the existing index, this operation requires additional storage roughly equal to the size of the existing index.

DML Behavior

Because this procedure is performed online, DML on the base table are permitted during this operation, and are processed as normal. All DML statements that occur during `RECREATE_INDEX_ONLINE` are logged into an online pending queue.

Towards the end of the recreate operation, there will be a short duration when DML will fail with an error being raised stating that the index is in an in-progress status. DML may hang again during the process, and the duration will depend on how many DML are logged in the online pending queue since the start of the recreate process.

Note that after the recreate index operation is complete, new information, from all the DML that becomes pending since `RECREATE_INDEX_ONLINE` started, may not be immediately reflected. As with creating an index with `INDEXTYPE IS ctxsys.context ONLINE`, the index should be synchronized after the recreate index operation is complete, to bring it fully up-to-date.

> **See Also:**
>
> CTX_DDL.CREATE_SHADOW_INDEX and CTX_DDL.EXCHANGE_SHADOW_INDEX for information about how to manually go through each stage of recreation, and to schedule each step to run at a preferred time
>
> The `ONLINE` parameter under "Syntax for CONTEXT Index Type" on page 1-37

Sync and Optimize Behavior

Syncs issued against the index during the recreate operation are processed against the old, existing data. Syncs are also blocked during the same window when queries return errors. Optimize commands issued against the index during the recreate operation return immediately without error and without processing.

Query Behavior

During the recreate operation, the index can be queried normally most of the time. Queries return results based on the existing index and policy (or metadata) until after the final swap.

There is a short interval towards the end of RECREATE_INDEX_ONLINE when queries will return an error indicating that the column is not indexed. This duration should be short for regular queries. It is mainly the time taken for swapping data segments of the shadow index tables and the index tables, plus the time to delete all the rows in the pending queue. This is the same window of time when DML will fail.

During RECREATE_INDEX_ONLINE, if you issue DML statements and synchronize them, then you will be able to see the new rows when you query on the existing index. However, after RECREATE_INDEX_ONLINE finishes (swapping completes and query is on the new index) and before sync is performed, it is possible that you will not be able to query on the new rows, which once could be queried on the old index.

Note: Transactional queries are not supported.

Using RECREATE_INDEX_ONLINE with Local Partitioned Indexes

If the index is local partitioned, you cannot recreate index in one step. You must first create a shadow policy, and then run this procedure for every partition. You can specify SWAP or NOSWAP to indicate whether RECREATE_INDEX_ONLINE partition will swap the index partition data and index partition metadata or not. If the partition was built with NOSWAP, then another call to EXCHANGE_SHADOW_INDEX must be invoked later against this partition.

This procedure can also be used to update the metadata (for example, storage preference) of each partition when you specify NOPOPULATE in the parameter string. This is useful for incremental building of a shadow index through time-limited sync.

If NOPOPULATE is specified, then NOSWAP is silently enforced.

NOSWAP Behavior

During the recreate of the index partition, since no swapping is performed, queries on the partition are processed regularly. Until the swapping stage is reached, queries spanning multiple partitions return consistent results across partitions.

DML and sync are processed normally. Running optimize on partitions that are being recreated, or that have been built (but not swapped), simply returns without doing anything. Running optimize on a partition that has not been rebuilt processes normally.

As with a global index, when all of the partitions use NOSWAP, the additional storage requirement is roughly equal to the size of the existing index.

SWAP Behavior

Because index partition data and metadata are swapped after index recreate, queries that span multiple partitions will not return consistent results from partition to partition, but will always be correct with respect to each index partition. There is also a short interval towards the end of partition recreate, when the index partition is swapped, during which a query will return a "column not indexed" error.

When partitions are recreated with SWAP, the additional storage requirement for the operation is equal to the size of the existing index partition.

DML on the partition is blocked. Sync is also blocked during swapping.

Related Topics

See also CREATE_SHADOW_INDEX on page 7-41 and DROP_SHADOW_INDEX on page 7-49, and EXCHANGE_SHADOW_INDEX on page 7-51, as well as *Oracle Text Application Developer's Guide*.

REMOVE_INDEX

Removes the index with the specified column list from a CTXCAT index set preference.

> **Note:** This procedure does not remove a CTXCAT sub-index from the existing index. To do so, you must drop your index and re-index with the modified index set preference.

Syntax

```
CTX_DDL.REMOVE_INDEX(
    set_name       IN VARCHAR2,
     column_list    IN VARCHAR2
     language       IN VARCHAR2 default NULL
);
```

set_name
Specify the name of the index set.

column_list
Specify the name of the column list to remove.

REMOVE_MDATA

Use this procedure to remove metadata values, which are associated with an MDATA section, from a document. Only the owner of the index is allowed to call ADD_MDATA and REMOVE_MDATA.

Syntax

```
CTX_DDL.REMOVE_MDATA(
     idx_name           IN VARCHAR2,
     section_name       IN VARCHAR2,
     values             SYS.ODCIVARCHAR2LIST,
     rowids             SYS.ODCIRIDLIST,
     [part_name]        IN VARCHAR2]
);
```

idx_name
Name of the text index that contains the named *rowids*.

section_name
Name of the MDATA section.

values
List of metadata values. If a metadata value contains a comma, the comma must be escaped with a backslash.

rowids
Rowids from which to remove the metadata values.

[part_name]
Name of the index partition, if any. Must be provided for local partitioned indexes and must be NULL for global, non-partitioned indexes.

Example

This example removes the MDATA value *blue* from the MDATA section BGCOLOR.

```
ctx_ddl.remove_mdata('idx_docs', 'bgcolor', 'blue', 'rows');
```

Related Topics

See also "ADD_MDATA" on page 7-9; "ADD_MDATA_SECTION" on page 7-12; "MDATA" on page 3-28; as well as the Section Searching chapter of *Oracle Text Application Developer's Guide*.

Notes

These updates are updates directly on the index itself, not on the actual contents stored in the base table. Therefore, they will not survive when the Text index is rebuilt.

REMOVE_SECTION

The REMOVE_SECTION procedure removes the specified section from the specified section group. You can specify the section by name or ID. View section ID with the CTX_USER_SECTIONS view.

Syntax 1

Use the following syntax to remove a section by section name:

```
CTX_DDL.REMOVE_SECTION(
  group_name       in    varchar2,
  section_name     in    varchar2
);
```

group_name
Specify the name of the section group from which to delete section_name.

section_name
Specify the name of the section to delete from group_name.

Syntax 2

Use the following syntax to remove a section by section ID:

```
CTX_DDL.REMOVE_SECTION(
  group_name     in    varchar2,
  section_id     in    number
);
```

group_name
Specify the name of the section group from which to delete section_id.

section_id
Specify the section ID of the section to delete from group_name.

Example

The following example drops a section called Title from the htmgroup:

```
begin
ctx_ddl.remove_section('htmgroup', 'Title');
end;
```

Related Topics

"ADD_FIELD_SECTION" on page 7-4

"ADD_SPECIAL_SECTION" on page 7-18

"ADD_ZONE_SECTION" on page 7-28

REMOVE_STOPCLASS

Removes a stopclass from a stoplist.

Syntax

```
CTX_DDL.REMOVE_STOPCLASS(
  stoplist_name  in   varchar2,
  stopclass      in   varchar2
);
```

stoplist_name
Specify the name of the stoplist.

stopclass
Specify the name of the stopclass to be removed.

Example

The following example removes the stopclass `NUMBERS` from the stoplist `mystop`.

```
begin
ctx_ddl.remove_stopclass('mystop', 'NUMBERS');
end;
```

Related Topics

"ADD_STOPCLASS" on page 7-20

REMOVE_STOPTHEME

Removes a stoptheme from a stoplist.

Syntax

```
CTX_DDL.REMOVE_STOPTHEME(
  stoplist_name   in   varchar2,
  stoptheme       in   varchar2
);
```

stoplist_name
Specify the name of the stoplist.

stoptheme
Specify the stoptheme to be removed from `stoplist_name`.

Example

The following example removes the stoptheme *banking* from the stoplist `mystop`:

```
begin
ctx_ddl.remove_stoptheme('mystop', 'banking');
end;
```

Related Topics

"ADD_STOPTHEME" on page 7-23

REMOVE_STOPWORD

Removes a stopword from a stoplist. To have the removal of a stopword be reflected in the index, you must rebuild your index.

Syntax

```
CTX_DDL.REMOVE_STOPWORD(
stoplist_name  in   varchar2,
stopword       in   varchar2,
language       in   varchar2 default NULL
);
```

stoplist_name
Specify the name of the stoplist.

stopword
Specify the stopword to be removed from `stoplist_name`.

language
Specify the language of `stopword` to remove when the stoplist you specify with `stoplist_name` is of type `MULTI_STOPLIST`. You must specify the globalization support name or abbreviation of an Oracle Text-supported language. You can also remove `ALL` stopwords.

Example

The following example removes a stopword *because* from the stoplist `mystop`:

```
begin
ctx_ddl.remove_stopword('mystop','because');
end;
```

Related Topics

"ADD_STOPWORD" on page 7-24

REMOVE_SUB_LEXER

Removes a sub-lexer from a multi-lexer preference. You cannot remove the lexer for `DEFAULT`.

Syntax

```
CTX_DDL.REMOVE_SUB_LEXER(
lexer_name  in   varchar2,
language    in   varchar2 default NULL
);
```

lexer_name
Specify the name of the multi-lexer preference.

language
Specify the language of the sub-lexer to remove. You must specify the globalization support name or abbreviation of an Oracle Text-supported language.

Example

The following example removes a sub-lexer *german_lexer* of language `german`:

```
begin
ctx_ddl.remove_sub_lexer('german_lexer','german');
end;
```

Related Topics

on page 7-26 on page 7-26

REPLACE_INDEX_METADATA

Use this procedure to replace metadata in local domain indexes at the global (index) level.

> **Note:** The `ALTER INDEX PARAMETERS` command performs the same function as this procedure and can replace more than just metadata. For that reason, using `ALTER INDEX PARAMETERS` is the preferred method of replacing metadata at the global (index) level and should be used in place of this procedure when possible. For more information, see "ALTER INDEX PARAMETERS Syntax" on page 1-3.
>
> `CTX_REPLACE_INDEX_METADATA` may be deprecated in a future release of Oracle Text.

Syntax

```
CTX_DDL.REPLACE_INDEX_METADATA(idx_name IN VARCHAR2,
                      parameter_string IN VARCHAR2);
```

idx_name
Specify the name of the index whose metadata you want to replace.

parameter_string
Specify the parameter string to be passed to `ALTER INDEX`. This must begin with `'REPLACE METADATA'`.

Notes

`ALTER INDEX REBUILD PARAMETERS ('REPLACE METADATA')` does not work for a local partitioned index at the index (global) level; you cannot, for example, use that `ALTER INDEX` syntax to change a global preference, such as filter or lexer type, without rebuilding the index. Therefore, `CTX_DDL.REPLACE_INDEX_METADATA` is provided as a method of overcoming this limitation of `ALTER INDEX`.

Though it is meant as a way to replace metadata for a local partitioned index, `CTX_DDL.REPLACE_INDEX_METADATA` can be used on a global, non-partitioned index, as well.

`REPLACE_INDEX_METADATA` cannot be used to change the sync type at the partition level; that is, *parameter_string* cannot be `'REPLACE METADATA SYNC'`. For that purpose, use `ALTER INDEX REBUILD PARTITION` to change the sync type at the partition level.

Related Topics

See also "ALTER INDEX PARAMETERS Syntax" on page 1-3 and "ALTER INDEX REBUILD Syntax" on page 1-4.

SET_ATTRIBUTE

Sets a preference attribute. Use this procedure after you have created a preference with CTX_DDL.CREATE_PREFERENCE.

Syntax

```
CTX_DDL.SET_ATTRIBUTE(preference_name IN VARCHAR2,
                      attribute_name  IN VARCHAR2,
                      attribute_value IN VARCHAR2);
```

preference_name
Specify the name of the preference.

attribute_name
Specify the name of the attribute.

attribute_value
Specify the attribute value. Specify boolean values as `TRUE` or `FALSE`, `T` or `F`, `YES` or `NO`, `Y` or `N`, `ON` or `OFF`, or `1` or `0`.

Example

Specifying File Data Storage

The following example creates a data storage preference called `filepref` that tells the system that the files to be indexed are stored in the operating system. The example then uses CTX_DDL.SET_ATTRIBUTE to set the `PATH` attribute to the directory `/docs`.

```
begin
ctx_ddl.create_preference('filepref', 'FILE_DATASTORE');
ctx_ddl.set_attribute('filepref', 'PATH', '/docs');
end;
```

> **See Also:** For more information about data storage, see "Datastore Types" on page 2-2
>
> For more examples of using `SET_ATTRIBUTE`, see "CREATE_PREFERENCE" on page 7-35

SYNC_INDEX

Synchronizes the index to process inserts, updates, and deletes to the base table.

> **Note:** Because CTX_DDL.SYNC_INDEX issues implicit commits, calling CTX_DDL.SYNC_INDEX in a trigger is strongly discouraged. Doing so can result in errors being raised, as both SYNC_INDEX and post-commit $R LOB maintenance try to update the same $R LOB.

Syntax

```
CTX_DDL.SYNC_INDEX(
idx_name       IN  VARCHAR2 DEFAULT NULL
memory          IN VARCHAR2 DEFAULT NULL,
part_name       IN VARCHAR2 DEFAULT NULL,
parallel_degree IN NUMBER DEFAULT 1
maxtime         IN NUMBER DEFAULT NULL,
locking         IN NUMBER DEFAULT LOCK_WAIT
);
```

idx_name

Specify the name of the index to synchronize.

> **Note:** When idx_name is null, all CONTEXT, CTXRULE, and CTXXPATH indexes that have pending changes are synchronized. You must be connected as ctxsys to perform this operation. Each index or index partition is synchronized in sequence, one after the other. Because of this, the individual syncs are performed with locking set to NOWAIT and maxtime set to 0. Any values that you specify for locking or maxtime on the SYNC_INDEX call are ignored. However, the memory and parallel_degree parameters are passed on to the individual synchronizations.

memory

Specify the runtime memory to use for synchronization. This value overrides the DEFAULT_INDEX_MEMORY system parameter.

The memory parameter specifies the amount of memory Oracle Text uses for the synchronization operation before flushing the index to disk. Specifying a large amount of memory:

- Improves indexing performance because there is less I/O
- Improves query performance and maintenance because there is less fragmentation
- The indexing memory size specified in the second argument applies to each parallel slave. For exmaple, if the memory argument is set to 500M and parallel_degree is set to 2, then ensure that there is at least 1GB of memory available on the system used for the parallel SYNC_INDEX.

Specifying smaller amounts of memory increases disk I/O and index fragmentation, but might be useful when runtime memory is scarce.

part_name
If your index is a local index, then you must specify the name of the index partition to synchronize otherwise an error is returned.

If your index is a global, non-partitioned index, then specify NULL, which is the default.

parallel_degree
Specify the degree to run parallel synchronize. A number greater than 1 turns on parallel synchronize. The actual degree of parallelism might be smaller depending on your resources.

maxtime
Indicate a suggested time limit on the operation, in minutes. SYNC_INDEX will process as many documents in the queue as possible within the time limit. The maxtime value of NULL is equivalent to CTX_DDL.MAXTIME_UNLIMITED. This parameter is ignored when SYNC_INDEX is invoked without an index name, in which case maxtime value of 0 is used instead. The locking parameter is ignored for automatic syncs (that is, SYNC ON COMMIT or SYNC EVERY).

The time limit specified is treated as approximate. The actual time taken may be somewhat less than or greater than what you specify. The "time clock" for maxtime does not start until the SYNC lock is acquired.

locking
Configure how SYNC_INDEX deals with the situation where another sync is already running on the same index or index partition. When locking is ignored because SYNC_INDEX is invoked without an index name, then locking value of LOCK_NOWAIT is used instead. The locking parameter is ignored for automatic syncs (that is, SYNC ON COMMIT or SYNC EVERY).

The options for locking are:

CTX_DDL.LOCK_WAIT	If another sync is running, wait until the running sync is complete, then begin sync. (In the event of not being able to get a lock, it will wait forever and ignore the maxtime setting.)
CTX_DDL.LOCK_NOWAIT	If another sync is running, immediately returns without error.
CTX_DDL.LOCK_NOWAIT_ERROR	If another sync is running, error "DRG-51313: timeout while waiting for DML or optimize lock" is raised.

Example

The following example synchronizes the index myindex with 2 megabytes of memory:

```
begin
ctx_ddl.sync_index('myindex', '2M');
end;
```

The following example synchronizes the part1 index partition with 2 megabytes of memory:

```
begin
ctx_ddl.sync_index('myindex', '2M', 'part1');
end;
```

Notes

You can run CTX_DDL.SYNC_INDEX and CTX_DDL.OPTIMIZE_INDEX at the same time. You can also run CTX_DDL.SYNC_INDEX and CTX_DDL.OPTIMIZE_INDEX with parallelism at the same time. However, you should not run CTX_DDL.SYNC_INDEX with parallelism at the same time as CTX_DDL.OPTIMIZE_INDEX, nor CTX_DDL.SYNC_INDEX with parallelism at the same time as CTX_DDL.OPTIMIZE_INDEX with parallelism. If you should run one of these combinations, no error is generated; however, one operation will wait until the other is done.

Related Topics

"ALTER INDEX" on page 1-2

UNSET_ATTRIBUTE

Removes a set attribute from a preference.

Syntax

```
CTX_DDL.UNSET_ATTRIBUTE(preference_name varchar2,
                        attribute_name  varchar2);
```

preference_name
Specify the name of the preference.

attribute_name
Specify the name of the attribute.

Example

Enabling/Disabling Alternate Spelling

The following example shows how you can enable alternate spelling for German and disable alternate spelling with `CTX_DDL.UNSET_ATTRIBUTE`:

```
begin
ctx_ddl.create_preference('GERMAN_LEX', 'BASIC_LEXER');
ctx_ddl.set_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING', 'GERMAN');
end;
```

To disable alternate spelling, use the `CTX_DDL.UNSET_ATTRIBUTE` procedure as follows:

```
begin
ctx_ddl.unset_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING');
end;
```

Related Topics

"SET_ATTRIBUTE" on page 7-73

UPDATE_POLICY

Updates a policy created with CREATE_POLICY. Replaces the preferences of the policy. Null arguments are not replaced.

Syntax

```
CTX_DDL.UPDATE_POLICY(
         policy_name     IN VARCHAR2,
         filter          IN VARCHAR2 DEFAULT NULL,
         section_group   IN VARCHAR2 DEFAULT NULL,
         lexer           IN VARCHAR2 DEFAULT NULL,
         stoplist        IN VARCHAR2 DEFAULT NULL,
         wordlist        IN VARCHAR2 DEFAULT NULL);
```

policy_name
Specify the name of the policy to update.

filter
Specify the filter preference to use.

section_group
Specify the section group to use.

lexer
Specify the lexer preference to use.

stoplist
Specify the stoplist to use.

wordlist
Specify the wordlist to use.

8

CTX_DOC Package

This chapter describes the CTX_DOC PL/SQL package for requesting document services, such as highlighting extracted text or generating a list of themes for a document.

Many of these procedures exist in two versions: those that make use of indexes, and those that do not. Those that do not are called "policy-based" procedures. They are offered because there are times when you may want to use document services on a single document without creating a Context index in advance. Policy-based procedures enable you to do this.

The policy_* procedures mirror the conventional in-memory document services and are used with policy_name replacing index_ name, and document of type VARCHAR2, CLOB, BLOB, or BFILE replacing textkey. Thus, you need not create an index to obtain document services output with these procedures.

For the procedures that generate character offsets and lengths, such as HIGHLIGHT and TOKENS, Oracle Text follows USC-2 codepoint semantics.

The CTX_DOC package includes the following procedures and functions:

Name	Description
FILTER	Generates a plain text or HTML version of a document.
GIST	Generates a Gist or theme summaries for a document.
HIGHLIGHT	Generates plain text or HTML highlighting offset information for a document.
IFILTER	Generates a plain text version of binary data. Can be called from a USER_DATASTORE procedure.
MARKUP	Generates a plain text or HTML version of a document with query terms highlighted.
PKENCODE	Encodes a composite textkey string (value) for use in other CTX_DOC procedures.
POLICY_FILTER	Generates a plain text or HTML version of a document, without requiring an index.
POLICY_GIST	Generates a Gist or theme summaries for a document, without requiring an index.
POLICY_HIGHLIGHT	Generates plain text or HTML highlighting offset information for a document, without requiring an index.
POLICY_MARKUP	Generates a plain text or HTML version of a document with query terms highlighted, without requiring an index.

Name	Description
POLICY_SNIPPET	Generates a concordance for a document, based on query terms, without requiring an index.
POLICY_THEMES	Generates a list of themes for a document, without requiring an index.
POLICY_TOKENS	Generates all index tokens for a document, without requiring an index.
SET_KEY_TYPE	Sets `CTX_DOC` procedures to accept rowid or primary key document identifiers.
SNIPPET	Generates a concordance for a document, based on query terms.
THEMES	Generates a list of themes for a document.
TOKENS	Generates all index tokens for a document.

FILTER

Use the CTX_DOC.FILTER procedure to generate either a plain text or HTML version of a document. You can store the rendered document in either a result table or in memory. This procedure is generally called after a query, from which you identify the document to be filtered.

Note: The resultant HTML document does not include graphics.

Syntax 1: In-memory Result Storage

```
exec CTX_DOC.FILTER(
          index_name  IN VARCHAR2,
          textkey     IN VARCHAR2,
          restab      IN OUT NOCOPY CLOB,
          plaintext   IN BOOLEAN  DEFAULT FALSE);

exec  CTX_DOC.HIGHLIGHT_CLOB_QUERY(
          index_name IN VARCHAR2,
          textkey IN VARCHAR2,
          text_query IN CLOB,
          restab IN OUT NOCOPY HIGHLIGHT_TAB,
          plaintext IN BOOLEAN DEFAULT FALSE);
```

Syntax 2: Result Table Storage

```
exec CTX_DOC.FILTER(
          index_name  IN VARCHAR2,
          textkey     IN VARCHAR2,
          restab      IN VARCHAR2,
          query_id    IN NUMBER DEFAULT 0,
          plaintext   IN BOOLEAN  DEFAULT FALSE);

exec  CTX_DOC.HIGHLIGHT_CLOB_QUERY(
          index_name IN VARCHAR2,
          textkey IN VARCHAR2,
          text_query IN CLOB,
          restab IN VARCHAR2,
          query_id IN NUMBER DEFAULT 0,
          plaintext IN BOOLEAN DEFAULT FALSE);
```

index_name

Specify the name of the index associated with the text column containing the document identified by textkey.

textkey

Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can be as follows:

- a single column primary key value
- encoded specification for a composite (multiple column) primary key. Use CTX_DOC.PKENCODE
- the rowid of the row containing the document

Toggle between primary key and rowid identification using CTX_DOC.SET_KEY_TYPE.

restab
You can specify that this procedure store the marked-up text to either a table or to an in-memory CLOB.

To store results to a table, specify the name of the table. The table to which you want to store results must exist before you make this call.

> **See Also:** "Filter Table" in Appendix A, "Oracle Text Result Tables" for more information about the structure of the filter result table

To store results in memory, specify the name of the CLOB locator. If restab is NULL, then a temporary CLOB is allocated and returned. You must de-allocate the locator after using it with DBMS_LOB.FREETEMPORARY().

If restab is not NULL, then the CLOB is truncated before the operation.

query_id
Specify an identifier to use to identify the row inserted into restab.

When query_id is not specified or set to NULL, it defaults to 0. You must manually truncate the table specified in restab.

plaintext
Specify TRUE to generate a plaintext version of the document. Specify FALSE to generate an HTML version of the document if you are using the AUTO_FILTER filter or indexing HTML documents.

Example

In-Memory Filter

The following code shows how to filter a document to HTML in memory.

```
declare
mklob clob;
amt number := 40;
line varchar2(80);

begin
 ctx_doc.filter('myindex','1', mklob, FALSE);
 -- mklob is NULL when passed-in, so ctx-doc.filter will allocate a temporary
 -- CLOB for us and place the results there.
 dbms_lob.read(mklob, amt, 1, line);
 dbms_output.put_line('FIRST 40 CHARS ARE:'||line);
 -- have to de-allocate the temp lob
 dbms_lob.freetemporary(mklob);
 end;
```

Create the filter result table to store the filtered document as follows:

```
create table filtertab (query_id  number,
                        document  clob);
```

To obtain a plaintext version of document with textkey 20, enter the following statement:

```
begin
```

```
ctx_doc.filter('newsindex', '20', 'filtertab', '0', TRUE);
end;
```

GIST

Use the CTX_DOC.GIST procedure to generate gist and theme summaries for a document. You can generate paragraph-level or sentence-level gists or theme summaries.

> **Note:** CTX_DOC.GIST requires an installed knowledge base. A knowledge base may or may not have been installed with Oracle Text. For more information on knowledge bases, see the *Oracle Text Application Developer's Guide*.

Syntax 1: In-Memory Storage

```
CTX_DOC.GIST(
index_name    IN VARCHAR2,
textkey       IN VARCHAR2,
restab        IN OUT CLOB,
glevel        IN VARCHAR2 DEFAULT 'P',
pov           IN VARCHAR2 DEFAULT 'GENERIC',
numParagraphs IN NUMBER DEFAULT 16,
maxPercent    IN NUMBER DEFAULT 10,
num_themes   IN NUMBER DEFAULT 50);
```

Syntax 2: Result Table Storage

```
CTX_DOC.GIST(
index_name    IN VARCHAR2,
textkey       IN VARCHAR2,
restab        IN VARCHAR2,
query_id      IN NUMBER DEFAULT 0,
glevel        IN VARCHAR2 DEFAULT 'P',
pov           IN VARCHAR2 DEFAULT NULL,
numParagraphs IN NUMBER DEFAULT 16,
maxPercent    IN NUMBER DEFAULT 10,
num_themes     IN NUMBER DEFAULT 50);
```

index_name

Specify the name of the index associated with the text column containing the document identified by textkey.

textkey

Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can as follows:

- a single column primary key value
- an encoded specification for a composite (multiple column) primary key. To encode a composite textkey, use the CTX_DOC.PKENCODE procedure
- the rowid of the row containing the document

Toggle between primary key and rowid identification using CTX_DOC.SET_KEY_TYPE.

restab

Specify that this procedure store the gist and theme summaries to either a table or to an in-memory CLOB.

To store results to a table specify the name of an existing table.

> **See Also:** "Gist Table" in Appendix A, "Oracle Text Result Tables"

To store results in memory, specify the name of the CLOB locator. If restab is NULL, then a temporary CLOB is allocated and returned. You must de-allocate the locator after using it.

If restab is not NULL, then the CLOB is truncated before the operation.

query_id
Specify an identifier to use to identify the row(s) inserted into restab.

glevel
Specify the type of gist or theme summary to produce. The possible values are:

- *P* for paragraph
- *S* for sentence

The default is *P*.

pov
Specify whether a gist or a single theme summary is generated. The type of gist or theme summary generated (sentence-level or paragraph-level) depends on the value specified for glevel.

To generate a gist for the entire document, specify a value of 'GENERIC' for pov. To generate a theme summary for a single theme in a document, specify the theme as the value for pov.

When using result table storage, if you do not specify a value for pov, then this procedure returns the generic gist plus up to 50 theme summaries for the document.

When using in-memory result storage to a CLOB, you must specify a pov. However, if you do not specify a pov, then this procedure generates only a generic gist for the document.

Note: The pov parameter is case sensitive. To return a gist for a document, specify 'GENERIC' in all uppercase. To return a theme summary, specify the theme exactly as it is generated for the document.

Only the themes generated by THEMES for a document can be used as input for pov.

---

numParagraphs
Specify the maximum number of document paragraphs (or sentences) selected for the document gist or theme summaries. The default is 16.

Note: The numParagraphs parameter is used only when this parameter yields a smaller gist or theme summary size than the gist or theme summary size yielded by the maxPercent parameter.

This means that the system always returns the smallest size gist or theme summary.

---

maxPercent
Specify the maximum number of document paragraphs (or sentences) selected for the document gist or theme summaries as a percentage of the total paragraphs (or sentences) in the document. The default is 10.

> **Note:** The `maxPercent` parameter is used only when this parameter yields a smaller gist or theme summary size than the gist or theme summary size yielded by the `numParagraphs` parameter.
>
> This means that the system always returns the smallest size gist or theme summary.

num_themes
Specify the number of theme summaries to produce when you do not specify a value for `pov`. For example, if you specify 10, this procedure returns the top 10 theme summaries. The default is 50.

If you specify 0 or NULL, then this procedure returns all themes in a document. If the document contains more than 50 themes, only the top 50 themes show conceptual hierarchy.

Examples

In-Memory Gist

The following example generates a non-default size generic gist of at most 10 paragraphs. The result is stored in memory in a `CLOB` locator. The code then de-allocates the returned `CLOB` locator after using it.

```
set serveroutput on;
declare
  gklob clob;
  amt number := 40;
  line varchar2(80);

begin
 ctx_doc.gist('newsindex','34',gklob, pov => 'GENERIC',numParagraphs => 10);
  -- gklob is NULL when passed-in, so ctx-doc.gist will allocate a temporary
  -- CLOB for us and place the results there.

  dbms_lob.read(gklob, amt, 1, line);
  dbms_output.put_line('FIRST 40 CHARS ARE:'||line);
  -- have to de-allocate the temp lob
  dbms_lob.freetemporary(gklob);
 end;
```

Result Table Gists

The following example creates a gist table called `CTX_GIST`:

```
create table CTX_GIST (query_id  number,
                       pov       varchar2(80),
                       gist      CLOB);
```

Gists and Theme Summaries

The following example returns a default sized paragraph-level gist for document 34 as well as the top 10 theme summaries in the document:

```
begin
   ctx_doc.gist('newsindex','34','CTX_GIST', 1, num_themes=>10);
end;
```

The following example generates a non-default size gist of at most 10 paragraphs:

```
begin
  ctx_doc.gist('newsindex','34','CTX_GIST',1,pov =>'GENERIC',numParagraphs=>10);
end;
```

The following example generates a gist whose number of paragraphs is at most 10 percent of the total paragraphs in document:

```
begin
  ctx_doc.gist('newsindex','34','CTX_GIST',1,pov => 'GENERIC',  maxPercent => 10);
end;
```

Theme Summary

The following example returns a paragraph-level theme summary for *insects* for document 34. The default theme summary size is returned.

```
begin
   ctx_doc.gist('newsindex','34','CTX_GIST',1, pov => 'insects');
end;
```

HIGHLIGHT

Use the CTX_DOC.HIGHLIGHT procedure to generate highlight offsets for a document. The offset information is generated for the terms in the document that satisfy the query you specify. These highlighted terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.

You can generate highlight offsets for either plaintext or HTML versions of the document. The table returned by CTX_DOC.HIGHLIGHT does not include any graphics found in the original document. Apply the offset information to the same documents filtered with CTX_DOC.FILTER.

You usually call this procedure after a query, from which you identify the document to be processed.

You can store the highlight offsets to either an in-memory PL/SQL table or a result table.

See CTX_DOC.POLICY_HIGHLIGHT on page 8-24 for a version of this procedure that does not require an index.

Syntax 1: In-Memory Result Storage

```
exec CTX_DOC.HIGHLIGHT(
        index_name  IN VARCHAR2,
        textkey     IN VARCHAR2,
        text_query  IN VARCHAR2,
        restab      IN OUT NOCOPY HIGHLIGHT_TAB,
        plaintext   IN BOOLEAN  DEFAULT FALSE);

exec CTX_DOC.HIGHLIGHT_CLOB_QUERY(
        index_name  IN VARCHAR2,
        textkey     IN VARCHAR2,
        text_query  IN CLOB,
        restab      IN OUT NOCOPY HIGHLIGHT_TAB,
        plaintext   IN BOOLEAN DEFAULT FALSE);
```

Syntax 2: Result Table Storage

```
exec CTX_DOC.HIGHLIGHT(
          index_name  IN VARCHAR2,
          textkey     IN VARCHAR2,
          text_query  IN VARCHAR2,
          restab      IN VARCHAR2,
          query_id    IN NUMBER   DEFAULT 0,
          plaintext   IN BOOLEAN  DEFAULT FALSE);

exec CTX_DOC.HIGHLIGHT_CLOB_QUERY(
          index_name  IN VARCHAR2,
          textkey     IN VARCHAR2,
          text_query  IN CLOB,
          restab      IN VARCHAR2,
          query_id    IN NUMBER DEFAULT 0,
          plaintext   IN BOOLEAN DEFAULT FALSE);
```

index_name
Specify the name of the index associated with the text column containing the document identified by textkey.

textkey
Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can be as follows:

- a single column primary key value
- encoded specification for a composite (multiple column) primary key. Use the CTX_DOC.PKENCODE procedure.
- the rowid of the row containing the document

Toggle between primary key and rowid identification using CTX_DOC.SET_KEY_TYPE.

text_query
Specify the original query expression used to retrieve the document. If NULL, no highlights are generated.

If `text_query` includes wildcards, stemming, fuzzy matching which result in stopwords being returned, `HIGHLIGHT` does not highlight the stopwords.

If `text_query` contains the threshold operator, the operator is ignored. The `HIGHLIGHT` procedure always returns highlight information for the entire result set.

restab
You can specify that this procedure store highlight offsets to either a table or to an in-memory PL/SQL table.

To store results to a table specify the name of the table. The table must exist before you call this procedure.

> **See Also:** "Highlight Table" in Appendix A, "Oracle Text Result Tables" for more information about the structure of the highlight result table.

To store results to an in-memory table, specify the name of the in-memory table of type `CTX_DOC.HIGHLIGHT_TAB`. The `HIGHLIGHT_TAB` datatype is defined as follows:

```
type highlight_rec is record (
  offset number,
  length number
);
type highlight_tab is table of highlight_rec index by binary_integer;
```

`CTX_DOC.HIGHLIGHT` clears `HIGHLIGHT_TAB` before the operation.

query_id
Specify the identifier used to identify the row inserted into restab.

When query_id is not specified or set to `NULL`, it defaults to 0. You must manually truncate the table specified in restab.

plaintext
Specify `TRUE` to generate a plaintext offsets of the document.

Specify `FALSE` to generate HTML offsets of the document if you are using the `AUTO_FILTER` filter or indexing HTML documents.

Examples

Create Highlight Table

Create the highlight table to store the highlight offset information:

```
create table hightab(query_id number,
                     offset number,
                     length number);
```

Word Highlight Offsets

To obtain HTML highlight offset information for document 20 for the word *dog*:

```
begin
ctx_doc.highlight('newsindex', '20', 'dog', 'hightab', 0, FALSE);
end;
```

Theme Highlight Offsets

Assuming the index *newsindex* has a theme component, obtain HTML highlight offset information for the theme query of *politics* by issuing the following query:

```
begin
ctx_doc.highlight('newsindex', '20', 'about(politics)', 'hightab', 0, FALSE);
end;
```

The output for this statement are the offsets to highlighted words and phrases that represent the theme of *politics* in the document.

Restrictions

`CTX_DOC.HIGHLIGHT` does not support the use of query templates or highlighting XML attribute values.

Related Topics

See Also: "POLICY_HIGHLIGHT" on page 8-24, "MARKUP" on page 8-14, and "SNIPPET" on page 8-36

IFILTER

Use this procedure to filter binary data to text.

This procedure takes binary data (BLOB IN), filters the data with the AUTO_FILTER filter, and writes the text version to a CLOB. (Any graphics in the original document are ignored.) CTX_DOC.IFILTER employs the safe callout, and it does not require an index, as CTX_DOC.FILTER does.

> **Note:** This procedure will not be supported in future releases. Applications should use CTX_DOC.POLICY_FILTER instead.

Requirements

Because CTX_DOC.IFILTER employs the safe callout mechanism, the SQL*Net listener must be running and configured for extproc agent startup.

Syntax

```
CTX_DOC.IFILTER(data IN BLOB, text IN OUT NOCOPY CLOB);
```

data
Specify the binary data to be filtered.

text
Specify the destination CLOB. The filtered data is placed in here. This parameter must be a valid CLOB locator that is writable. Passing NULL or a non-writable CLOB will result in an error. Filtered text will be appended to the end of existing content, if any.

Example

The document text used in a MATCHES query can be VARCHAR2 or CLOB. It does not accept BLOB input, so you cannot match filtered documents directly. Instead, you must filter the binary content to CLOB using the AUTO_FILTER filter. Assuming the document data is in bind variable :doc_blob:

```
declare
  doc_text clob;
begin
  -- create a temporary CLOB to hold the document text
  dbms_lob.createtemporary(doc_text, TRUE, DBMS_LOB.SESSION);

  -- call ctx_doc.ifilter to filter the BLOB to CLOB data
  ctx_doc.ifilter(:doc_blob, doc_text);

  -- now do the matches query using the CLOB version
  for c1 in (select * from queries where matches(query_string, doc_text)>0)
  loop
    -- do what you need to do here
  end loop;

  dbms_lob.freetemporary(doc_text);
end;
```

MARKUP

The CTX_DOC.MARKUP procedure takes a query specification and a document textkey and returns a version of the document in which the query terms are marked up. These marked-up terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.

You can set the marked-up output to be either plaintext or HTML. The marked-up document returned by CTX_DOC.MARKUP does not include any graphics found in the original document.

You can use one of the pre-defined tag sets for marking highlighted terms, including a tag sequence that enables HTML navigation.

You usually call CTX_DOC.MARKUP after a query, from which you identify the document to be processed.

You can store the marked-up document either in memory or in a result table.

See CTX_DOC.POLICY_MARKUP on page 8-26 for a version of this procedure that does not require an index.

Note: Oracle Text does not guarantee well-formed output from CTX.DOC.MARKUP, especially for terms that are already marked up with HTML or XML. In particular, unexpected nesting of markup tags may occasionally result.

Syntax 1: In-Memory Result Storage

```
exec CTX_DOC.MARKUP(
index_name      IN VARCHAR2,
textkey         IN VARCHAR2,
text_query      IN VARCHAR2,
restab          IN OUT NOCOPY CLOB,
plaintext       IN BOOLEAN   DEFAULT FALSE,
tagset          IN VARCHAR2  DEFAULT 'TEXT_DEFAULT',
starttag        IN VARCHAR2  DEFAULT NULL,
endtag          IN VARCHAR2  DEFAULT NULL,
prevtag         IN VARCHAR2  DEFAULT NULL,
nexttag         IN VARCHAR2  DEFAULT NULL);

exec CTX_DOC.MARKUP_CLOB_QUERY(
index_name      IN VARCHAR2,
textkey         IN VARCHAR2,
text_query      IN CLOB,
restab          IN OUT NOCOPY CLOB,
plaintext       IN BOOLEAN DEFAULT FALSE,
tagset          IN VARCHAR2 DEFAULT 'TEXT_DEFAULT',
starttag        IN VARCHAR2 DEFAULT NULL,
endtag          IN VARCHAR2 DEFAULT NULL,
prevtag         IN VARCHAR2 DEFAULT NULL,
nexttag         IN VARCHAR2 DEFAULT NULL);
```

Syntax 2: Result Table Storage

```
exec CTX_DOC.MARKUP(
index_name      IN VARCHAR2,
```

```
textkey         IN VARCHAR2,
text_query      IN VARCHAR2,
restab          IN VARCHAR2,
query_id        IN NUMBER    DEFAULT 0,
plaintext       IN BOOLEAN   DEFAULT FALSE,
tagset          IN VARCHAR2  DEFAULT 'TEXT_DEFAULT',
starttag        IN VARCHAR2  DEFAULT NULL,
endtag          IN VARCHAR2  DEFAULT NULL,
prevtag         IN VARCHAR2  DEFAULT NULL,
nexttag         IN VARCHAR2  DEFAULT NULL);

exec CTX_DOC.MARKUP_CLOB_QUERY(
index_name      IN VARCHAR2,
textkey         IN CLOB,
text_query      IN VARCHAR2,
restab          IN VARCHAR2,
query_id        IN NUMBER DEFAULT 0,
plaintext       IN BOOLEAN DEFAULT FALSE,
tagset          IN VARCHAR2 DEFAULT 'TEXT_DEFAULT',
starttag        IN VARCHAR2 DEFAULT NULL,
endtag          IN VARCHAR2 DEFAULT NULL,
prevtag         IN VARCHAR2 DEFAULT NULL,
nexttag         IN VARCHAR2 DEFAULT NULL);
```

index_name
Specify the name of the index associated with the text column containing the document identified by textkey.

textkey
Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can be as follows:

- A single column primary key value
- Encoded specification for a composite (multiple column) primary key. Use the CTX_DOC.PKENCODE procedure.
- The rowid of the row containing the document

Toggle between primary key and rowid identification using CTX_DOC.SET_KEY_TYPE.

text_query
Specify the original query expression used to retrieve the document.

If text_query includes wildcards, stemming, fuzzy matching which result in stopwords being returned, MARKUP does not highlight the stopwords.

If text_query contains the threshold operator, the operator is ignored. The MARKUP procedure always returns highlight information for the entire result set.

restab
You can specify that this procedure store the marked-up text to either a table or to an in-memory CLOB.

To store results to a table specify the name of the table. The result table must exist before you call this procedure.

> **See Also:** For more information about the structure of the markup result table, see "Markup Table" in Appendix A, "Oracle Text Result Tables".

To store results in memory, specify the name of the CLOB locator. If restab is NULL, a temporary CLOB is allocated and returned. You must de-allocate the locator after using it.

If restab is not NULL, the CLOB is truncated before the operation.

query_id
Specify the identifier used to identify the row inserted into restab.

When query_id is not specified or set to NULL, it defaults to 0. You must manually truncate the table specified in restab.

plaintext
Specify TRUE to generate plaintext marked-up document. Specify FALSE to generate a marked-up HTML version of document if you are using the AUTO_FILTER filter or indexing HTML documents.

tagset
Specify one of the following pre-defined tag sets. The second and third columns show how the four different tags are defined for each tagset:

Tagset	Tag	Tag Value
TEXT_DEFAULT	starttag	<<<
	endtag	>>>
	prevtag	
	nexttag	
HTML_DEFAULT	starttag	<B>
	endtag	</B>
	prevtag	
	nexttag	
HTML_NAVIGATE	starttag	<A NAME=ctx%CURNUM><B>
	endtag	</B></A>
	prevtag	<A HREF=#ctx%PREVNUM><</A>
	nexttag	<A HREF=#ctx%NEXTNUM>></A>

starttag
Specify the character(s) inserted by MARKUP to indicate the start of a highlighted term.

The sequence of starttag, endtag, prevtag and nexttag with respect to the highlighted word is as follows:

```
... prevtag starttag word endtag nexttag...
```

endtag
Specify the character(s) inserted by MARKUP to indicate the end of a highlighted term.

prevtag
Specify the markup sequence that defines the tag that navigates the user to the previous highlight.

In the markup sequences prevtag and nexttag, you can specify the following offset variables which are set dynamically:

Offset Variable	Value
%CURNUM	the current offset number
%PREVNUM	the previous offset number
%NEXTNUM	the next offset number

See the description of the HTML_NAVIGATE tag set for an example.

nexttag
Specify the markup sequence that defines the tag that navigates the user to the next highlight tag.

Within the markup sequence, you can use the same offset variables you use for prevtag. See the explanation for prevtag and the HTML_NAVIGATE tag set for an example.

Examples

In-Memory Markup

The following code takes document (*the dog chases the cat*), performs the assigned markup on it, and stores the result in memory.

```
set serveroutput on

drop table mark_tab;
create table mark_tab (id number primary key, text varchar2(80) );
insert into mark_tab values ('1', 'The dog chases the cat.');

create index mark_tab_idx on mark_tab(text)
        indextype is ctxsys.context parameters
        ('filter ctxsys.null_filter');

declare
mklob clob;
amt number := 40;
line varchar2(80);

begin
 ctx_doc.markup('mark_tab_idx','1','dog AND cat', mklob);
 -- mklob is NULL when passed-in, so ctx_doc.markup will
 -- allocate a temporary CLOB for us and place the results there.
 dbms_lob.read(mklob, amt, 1, line);
 dbms_output.put_line('FIRST 40 CHARS ARE:'||line);
 -- have to de-allocate the temp lob
 dbms_lob.freetemporary(mklob);
 end;
/
```

The output from this example shows what the marked-up document looks like:

```
FIRST 40 CHARS ARE:  The <<<dog>>> chases the <<<cat>>>.
```

Markup Table

Create the highlight markup table to store the marked-up document as follows:

```
create table markuptab (query_id  number,
                        document  clob);
```

Word Highlighting in HTML

You can also store your MARKUP results in a table. To create HTML highlight markup for the words *dog* or *cat* for document 23, enter the following statement:

```
begin
  ctx_doc.markup(index_name => 'my_index',
                      textkey => '23',
                      text_query => 'dog|cat',
                      restab => 'markuptab',
                      query_id => '1',
                      tagset => 'HTML_DEFAULT');
end;
```

Theme Highlighting in HTML

To create HTML highlight markup for the theme of *politics* for document 23, enter the following statement:

```
begin
  ctx_doc.markup(index_name => 'my_index',
                      textkey => '23',
                      text_query => 'about(politics)',
                      restab => 'markuptab',
                      query_id => '1',
                      tagset => 'HTML_DEFAULT');
end;
```

Restrictions

`CTX_DOC.MARKUP` does not support the use of query templates.

Related Topics

See Also: "POLICY_MARKUP" on page 8-26, "HIGHLIGHT" on page 8-10 and "SNIPPET" on page 8-36

PKENCODE

The CTX_DOC.PKENCODE function converts a composite textkey list into a single string and returns the string.

The string created by PKENCODE can be used as the primary key parameter textkey in other CTX_DOC procedures, such as CTX_DOC.THEMES and CTX_DOC.GIST.

Syntax

```
CTX_DOC.PKENCODE(
         pk1    IN VARCHAR2,
         pk2    IN VARCHAR2 DEFAULT NULL,
         pk4    IN VARCHAR2 DEFAULT NULL,
         pk5    IN VARCHAR2 DEFAULT NULL,
         pk6    IN VARCHAR2 DEFAULT NULL,
         pk7    IN VARCHAR2 DEFAULT NULL,
         pk8    IN VARCHAR2 DEFAULT NULL,
         pk9    IN VARCHAR2 DEFAULT NULL,
         pk10   IN VARCHAR2 DEFAULT NULL,
         pk11   IN VARCHAR2 DEFAULT NULL,
         pk12   IN VARCHAR2 DEFAULT NULL,
         pk13   IN VARCHAR2 DEFAULT NULL,
         pk14   IN VARCHAR2 DEFAULT NULL,
         pk15   IN VARCHAR2 DEFAULT NULL,
         pk16   IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

pk1-pk16
Each PK argument specifies a column element in the composite textkey list. You can encode at most 16 column elements.

Returns

String that represents the encoded value of the composite textkey.

Example

```
begin
ctx_doc.gist('newsindex',CTX_DOC.PKENCODE('smith', 14), 'CTX_GIST');
end;
```

In this example, *smith* and *14* constitute the composite textkey value for the document.

POLICY_FILTER

Generates a plain text or an HTML version of a document. With this procedure, no CONTEXT index is required.

This procedure uses a trusted callout.

Syntax

```
ctx_doc.policy_filter(policy_name    in  VARCHAR2,
                      document       in [VARCHAR2|CLOB|BLOB|BFILE],
                      restab         in out nocopy CLOB,
                      plaintext      in BOOLEAN default FALSE,
                      language       in VARCHAR2 default NULL,
                      format         in VARCHAR2 default NULL,
                      charset        in VARCHAR2 default NULL);
```

policy_name
Specify the policy name created with CTX_DDL.CREATE_POLICY.

document
Specify the document to filter.

restab
Specify the name of the CLOB locator.

plaintext
Specify TRUE to generate a plaintext version of the document. Specify FALSE to generate an HTML version of the document if you are using the AUTO_FILTER filter or indexing HTML documents.

language
Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See BASIC_LEXER on page 2-31 in Chapter 2, "Oracle Text Indexing Elements".

format
Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description in CREATE INDEX on page 1-36 in Chapter 1, "Oracle Text SQL Statements and Operators".

charset
Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table. See Indexing Mixed-Character Set Columns on page 2-18 in Chapter 2, "Oracle Text Indexing Elements".

POLICY_GIST

Generates a Gist or theme summary for document.You can generate paragraph-level or sentence-level gists or theme summaries. With this procedure, no CONTEXT index is required.

> **Note:** CTX_DOC.POLICY_GIST requires an installed knowledge base. A knowledge base may or may not have been installed with Oracle Text. For more information on knowledge bases, see *Oracle Text Application Developer's Guide*.

Syntax

```
ctx_doc.policy_gist(policy_name      in VARCHAR2,
                    document         in [VARCHAR2|CLOB|BLOB|BFILE],
                    restab           in out nocopy CLOB,
                    glevel           in VARCHAR2 default 'P',
                    pov              in VARCHAR2 default 'GENERIC',
                    numParagraphs    in VARCHAR2 default NULL,
                    maxPercent       in NUMBER default NULL,
                    num_themes       in NUMBER default 50
                    language         in VARCHAR2 default NULL,
                    format           in VARCHAR2 default NULL,
                    charset          in VARCHAR2 default NULL
);
```

policy_name
Specify the policy name created with CTX_DDL.CREATE_POLICY.

document
Specify the document for which to generate the Gist or theme summary.

restab
Specify the name of the CLOB locator.

glevel
Specify the type of gist or theme summary to produce. The possible values are:

- *P* for paragraph
- *S* for sentence

The default is *P*.

pov
Specify whether a gist or a single theme summary is generated. The type of gist or theme summary generated (sentence-level or paragraph-level) depends on the value specified for glevel.

To generate a gist for the entire document, specify a value of 'GENERIC' for pov. To generate a theme summary for a single theme in a document, specify the theme as the value for pov.

When using result table storage and you do not specify a value for pov, this procedure returns the generic gist plus up to 50 theme summaries for the document.

Note: The `pov` parameter is case sensitive. To return a gist for a document, specify '`GENERIC`' in all uppercase. To return a theme summary, specify the theme *exactly* as it is generated for the document.

Only the themes generated by THEMES for a document can be used as input for `pov`.

---

numParagraphs
Specify the maximum number of document paragraphs (or sentences) selected for the document gist or theme summaries. The default is 16.

Note: The `numParagraphs` parameter is used only when this parameter yields a smaller gist or theme summary size than the gist or theme summary size yielded by the `maxPercent` parameter.

This means that the system always returns the smallest size gist or theme summary.

---

maxPercent
Specify the maximum number of document paragraphs (or sentences) selected for the document gist or theme summaries as a percentage of the total paragraphs (or sentences) in the document. The default is 10.

Note: The `maxPercent` parameter is used only when this parameter yields a smaller gist or theme summary size than the gist or theme summary size yielded by the `numParagraphs` parameter.

This means that the system always returns the smallest size gist or theme summary.

---

num_themes
Specify the number of theme summaries to produce when you do not specify a value for `pov`. For example, if you specify 10, this procedure returns the top 10 theme summaries. The default is 50.

If you specify 0 or NULL, this procedure returns all themes in a document. If the document contains more than 50 themes, only the top 50 themes show conceptual hierarchy.

language
Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See "MULTI_LEXER" on page 2-38.

format
Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description in "CREATE INDEX" on page 1-36.

charset
Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table.

POLICY_HIGHLIGHT

Generates plain text or HTML highlighting offset information for a document.With this procedure, no CONTEXT index is required.

The offset information is generated for the terms in the document that satisfy the query you specify. These highlighted terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.

You can generate highlight offsets for either plaintext or HTML versions of the document. You can apply the offset information to the same documents filtered with CTX_DOC.FILTER.

Syntax

```
exec ctx_doc.policy_highlight(
                        policy_name  in  VARCHAR2,
                        document     in  [VARCHAR2|CLOB|BLOB|BFILE],
                        text_query   in VARCHAR2,
                        restab       in out nocopy highlight_tab,
                        plaintext    in boolean FALSE
                        language     in VARCHAR2 default NULL,
                        format       in VARCHAR2 default NULL,
                        charset      in VARCHAR2 default NULL
);

exec ctx_doc.policy_highlight_clob_query(
                        policy_name  in VARCHAR2,
                        document     in [VARCHAR2|CLOB|BLOB|BFILE],
                        text_query   in CLOB,
                        restab       in out nocopy highlight_tab,
                        plaintext    in boolean FALSE
                        language     in VARCHAR2 default NULL,
                        format       in VARCHAR2 default NULL,
                        charset      in VARCHAR2 default NULL
);
```

policy_name
Specify the policy name created with CTX_DDL.CREATE_POLICY.

document
Specify the document to generate highlighting offset information.

text_query
Specify the original query expression used to retrieve the document. If NULL, no highlights are generated.

If text_query includes wildcards, stemming, or fuzzy matching which result in stopwords being returned, this procedure does not highlight the stopwords.

If text_query contains the threshold operator, the operator is ignored. This procedure always returns highlight information for the entire result set.

restab
Specify the name of the highlight_tab PL/SQL index-by-table type.

> **See Also:** see "HIGHLIGHT" on page 8-10 for more information about the structure of the highlight_tab table type.

plaintext
Specify `TRUE` to generate a plaintext offsets of the document.

Specify `FALSE` to generate HTML offsets of the document if you are using the `AUTO_FILTER` filter or indexing HTML documents.

language
Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See "MULTI_LEXER" in Chapter 2, "Oracle Text Indexing Elements".

format
Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description under "CREATE INDEX" on page 1-36.

charset
Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table.

Restrictions

`CTX_DOC.POLICY_HIGHLIGHT` does not support the use of query templates.

POLICY_MARKUP

Generates plain text or HTML version of a document with query terms highlighted. With this procedure, no CONTEXT index is required.

The CTX_DOC.POLICY_MARKUP procedure takes a query specification and a document and returns a version of the document in which the query terms are marked up. These marked-up terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.

You can set the marked-up output to be either plaintext or HTML.

You can use one of the pre-defined tag sets for marking highlighted terms, including a tag sequence that enables HTML navigation.

Syntax

```
ctx_doc.policy_markup(policy_name     in VARCHAR2,
                      document        in [VARCHAR2|CLOB|BLOB|BFILE],
                      text_query      in VARCHAR2,
                      restab          in out nocopy CLOB,
                      plaintext       in BOOLEAN default FALSE,
                      tagset          in VARCHAR2 default 'TEXT_DEFAULT',
                      starttag        in VARCHAR2 default NULL,
                      endtag          in VARCHAR2 default NULL,
                      prevtag         in VARCHAR2 default NULL,
                      nexttag         in VARCHAR2 default NULL
                      language        in VARCHAR2 default NULL,
                      format          in VARCHAR2 default NULL,
                      charset         in VARCHAR2 default NULL
);

ctx_doc.policy_markup_clob_query(
                      policy_name     in VARCHAR2,
                      document        in [VARCHAR2|CLOB|BLOB|BFILE],
                      text_query      in CLOB,
                      restab          in out nocopy CLOB,
                      plaintext       in BOOLEAN default FALSE,
                      tagset          in VARCHAR2 default 'TEXT_DEFAULT',
                      starttag        in VARCHAR2 default NULL,
                      endtag          in VARCHAR2 default NULL,
                      prevtag         in VARCHAR2 default NULL,
                      nexttag         in VARCHAR2 default NULL
                      language        in VARCHAR2 default NULL,
                      format          in VARCHAR2 default NULL,
                      charset         in VARCHAR2 default NULL
);
```

policy_name
Specify the policy name created with CTX_DDL.CREATE_POLICY.

document
Specify the document to generate highlighting offset information.

text_query
Specify the original query expression used to retrieve the document.

If text_query includes a NULL, then this procedure will fail and generate errors.

If text_query includes wildcards, stemming, or fuzzy matching which result in stopwords being returned, then this procedure does not highlight the stopwords.

If text_query contains the threshold operator, the operator is ignored. This procedure always returns highlight information for the entire result set.

restab
Specify the name of the CLOB locator.

plaintext
Specify TRUE to generate a plaintext marked-up document. Specify FALSE to generate a marked-up HTML version of the document if you are using the AUTO_FILTER filter or indexing HTML documents.

tagset
Specify one of the following pre-defined tag sets. The second and third columns show how the four different tags are defined for each tagset:

Tagset	Tag	Tag Value
TEXT_DEFAULT	starttag	<<<
	endtag	>>>
	prevtag	
	nexttag	
HTML_DEFAULT	starttag	<B>
	endtag	</B>
	prevtag	
	nexttag	
HTML_NAVIGATE	starttag	<A NAME=ctx%CURNUM><B>
	endtag	</B></A>
	prevtag	<A HREF=#ctx%PREVNUM><</A>
	nexttag	<A HREF=#ctx%NEXTNUM>></A>

starttag
Specify the character(s) inserted by MARKUP to indicate the start of a highlighted term.

The sequence of starttag, endtag, prevtag and nexttag with regard to the highlighted word is as follows:

```
... prevtag starttag word endtag nexttag...
```

endtag
Specify the character(s) inserted by MARKUP to indicate the end of a highlighted term.

prevtag
Specify the markup sequence that defines the tag that navigates the user to the previous highlight.

In the markup sequences prevtag and nexttag, you can specify the following offset variables which are set dynamically:

Offset Variable	Value
%CURNUM	the current offset number
%PREVNUM	the previous offset number
%NEXTNUM	the next offset number

See the description of the HTML_NAVIGATE tagset for an example.

nexttag
Specify the markup sequence that defines the tag that navigates the user to the next highlight tag.

Within the markup sequence, you can use the same offset variables you use for prevtag. See the explanation for prevtag and the HTML_NAVIGATE tagset for an example.

language
Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See "MULTI_LEXER" in Chapter 2, "Oracle Text Indexing Elements".

format
Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description in "CREATE INDEX".

charset
Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table. See "Indexing Mixed-Character Set Columns" in Chapter 2, "Oracle Text Indexing Elements".

Restrictions

CTX_DOC.POLICY_MARKUP does not support the use of query templates.

POLICY_SNIPPET

Display marked-up keywords in context. The returned text contains either the words that satisfy a word query or the themes that satisfy an ABOUT query. This version of the CTX_DOC.SNIPPET procedure does not require an index.

Syntax

Syntax 1

```
exec CTX_DOC.POLICY_SNIPPET(
policy_name              IN VARCHAR2,
document                 IN [VARCHAR2|CLOB|BLOB|BFILE],
text_query               IN VARCHAR2,
language                 IN VARCHAR2 default NULL,
format                   IN VARCHAR2 default NULL,
charset                  IN VARCHAR2 default NULL,
starttag                 IN VARCHAR2 DEFAULT '<b>',
endtag                   IN VARCHAR2 DEFAULT '</b>',
entity_translation       IN BOOLEAN  DEFAULT TRUE,
separator                IN VARCHAR2 DEFAULT '<b>...</b>'
)
return varchar2;
```

Syntax 2

```
exec CTX_DOC.POLICY_SNIPPET_CLOB_QUERY(
policy_name              IN VARCHAR2,
document                 IN [VARCHAR2|CLOB|BLOB|BFILE],
text_query               IN CLOB,
language                 IN VARCHAR2 default NULL,
format                   IN VARCHAR2 default NULL,
charset                  IN VARCHAR2 default NULL,
starttag                 IN VARCHAR2 DEFAULT '<b>',
endtag                   IN VARCHAR2 DEFAULT '</b>',
entity_translation       IN BOOLEAN DEFAULT TRUE,
separator                IN VARCHAR2 DEFAULT '<b>...</b>'
)
return varchar2;
```

policy_name
Specify the name of a policy created with CTX_DDL.CREATE_POLICY.

document
Specify the document in which to search for keywords.

text_query
Specify the original query expression used to retrieve the document. If NULL, no highlights are generated.

If text_query includes wildcards, stemming, fuzzy matching which result in stopwords being returned, POLICY_SNIPPET does not highlight the stopwords.

If text_query contains the threshold operator, the operator is ignored.

language
Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See MULTI_LEXER in Chapter 2, "Oracle Text Indexing Elements".

format
Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description in "CREATE INDEX".

charset
Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table. See "Indexing Mixed-Character Set Columns" in Chapter 2, "Oracle Text Indexing Elements".

starttag
Specify the start tag for marking up the query keywords. Default is '<*b*>'.

endtag
Specify the end tag for marking up the query keywords. Default is '</*b*>'.

entity_translation
Specify if you want HTML entities to be translated. The default is TRUE, which means the special entities (<, >, and &) are translated into their alternate forms ('*<*', '*>*', and '*&*') when output by the procedure. However, special characters in the markup tags generated by `CTX_DOC.POLICY_SNIPPET` will not be translated.

separator
Specify the string separating different returned fragments. Default is '<*b*>...</*b*>'.

Limitations

`CTX_DOC.POLICY_SNIPPET` does not support the use of query templates.

`CTX_DOC.POLICY_SNIPPET` displays marked-up keywords in context when used with `NULL_SECTION_GROUP`. However, there are limitations when using this procedure with XML documents. When used with `XML_SECTION_GROUP` or `AUTO_SECTION_GROUP`, the XML structure is ignored and user-specified tags are stripped out, which results in parts of surrounding text to be included in the returned snippet.

Related Topics

See Also: "SNIPPET" on page 8-36, "HIGHLIGHT" on page 8-10, and "MARKUP" on page 8-14

POLICY_THEMES

Generates a list of themes for a document. With this procedure, no CONTEXT index is required.

> **Note:** CTX_DOC.POLICY_THEMES requires an installed knowledge base. A knowledge base may or may not have been installed with Oracle Text. For more information on knowledge bases, see the *Oracle Text Application Developer's Guide*.

Syntax

```
ctx_doc.policy_themes(policy_name    in VARCHAR2,
                      document       in [VARCHAR2|CLOB|BLOB|BFILE],
                      restab         in out nocopy theme_tab,
                      full_themes    in BOOLEAN default FALSE,
                      num_themes     in number    default 50
                      language       in VARCHAR2 default NULL,
                      format         in VARCHAR2 default NULL,
                      charset        in VARCHAR2 default NULL
);
```

policy_name
Specify the policy you create with CTX_DDL.CREATE_POLICY.

document
Specify the document for which to generate a list of themes.

restab
Specify the name of the theme_tab PL/SQL index-by-table type.

> **See Also:** "THEMES" on page 8-39 for more information about the structure of the theme_tab type.

full_themes
Specify whether this procedure generates a single theme or a hierarchical list of parent themes (full themes) for each document theme.

Specify TRUE for this procedure to write full themes to the THEME column of the result table.

Specify FALSE for this procedure to write single theme information to the THEME column of the result table. This is the default.

num_themes
Specify the maximum number of themes to retrieve. For example, if you specify 10, up to first 10 themes are returned for the document. The default is 50.

If you specify 0 or NULL, this procedure returns all themes in a document. If the document contains more than 50 themes, only the first 50 themes show conceptual hierarchy.

language
Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See "MULTI_LEXER" in Chapter 2, "Oracle Text Indexing Elements".

format
Specify the format of the document. Use an Oracle Text supported format value, either `TEXT`, `BINARY` or `IGNORE` as you would specify in the format column of the base table. For more information, see the format column description in "CREATE INDEX" in Chapter 1, "Oracle Text SQL Statements and Operators".

charset
Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table. See "Indexing Mixed-Character Set Columns" in Chapter 2, "Oracle Text Indexing Elements".

Example

Create a policy:

```
exec ctx_ddl.create_policy('mypolicy');
```

Run themes:

```
declare
  la      varchar2(200);
  rtab    ctx_doc.theme_tab;
begin
   ctx_doc.policy_themes('mypolicy',
           'To define true madness, What is''t but to be nothing but mad?', rtab);
   for i in 1..rtab.count loop
     dbms_output.put_line(rtab(i).theme||':'||rtab(i).weight);
   end loop;
end;
```

POLICY_TOKENS

Generate all index tokens for document.With this procedure, no CONTEXT index is required.

Syntax

```
ctx_doc.policy_tokens(policy_name    in  VARCHAR2,
                      document       in  [VARCHAR2|CLOB|BLOB|BFILE],
                      restab         in  out nocopy token_tab,
                      language       in  VARCHAR2 default NULL,
                      format         in  VARCHAR2 default NULL,
                      charset        in  VARCHAR2 default NULL);
```

policy_name
Specify the policy name created with CTX_DDL.CREATE_POLICY.

document
Specify the document for which to generate tokens.

restab
Specify the name of the token_tab PL/SQL index-by-table type.

The tokens returned are those tokens which are inserted into the index for the document. Stop words are not returned. Section tags are not returned because they are not text tokens.

> **See Also:** "TOKENS" on page 8-42 of this chapter for more information about the structure of the token_tab type

language
Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See "MULTI_LEXER" in Chapter 2, "Oracle Text Indexing Elements".

format
Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description in "CREATE INDEX".

charset
Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table. See "Indexing Mixed-Character Set Columns" in Chapter 2, "Oracle Text Indexing Elements".

Example

Get tokens:

```
declare
  la     varchar2(200);
  rtab   ctx_doc.token_tab;
begin
   ctx_doc.policy_tokens('mypolicy',
        'To define true madness, What is''t but to be nothing but mad?',rtab);
   for i in 1..rtab.count loop
     dbms_output.put_line(rtab(i).offset||':'||rtab(i).token);
```

```
   end loop;
end;
```

SET_KEY_TYPE

Use this procedure to set the CTX_DOC procedures to accept either the ROWID or the PRIMARY_KEY document identifiers. This setting affects the invoking session only.

Syntax

```
ctx_doc.set_key_type(key_type in varchar2);
```

key_type
Specify either ROWID or PRIMARY_KEY as the input key type (document identifier) for CTX_DOC procedures.

This parameter defaults to the value of the CTX_DOC_KEY_TYPE system parameter.

> **Note:** When your base table has no primary key, setting key_type to PRIMARY_KEY is ignored. The textkey parameter that you specify for any CTX_DOC procedure is interpreted as a ROWID.

Example

The following example sets CTX_DOC procedures to accept primary key document identifiers.

```
begin
ctx_doc.set_key_type('PRIMARY_KEY');
end
```

SNIPPET

Use the CTX_DOC.SNIPPET procedure to produce a concordance for a document. A concordance is a text fragment that contains a query term with some of its surrounding text. This is also sometimes known as Key Word in Context or KWIC, because it returns query keywords marked up in their surrounding text, which enables the user to evaluate them in context. The returned text can also contain themes that satisfy an ABOUT query.

For example, a search on *brillig* and *slithey* might return one relevant fragment of a document as follows:

```
'Twas <b>brillig</b>, and the <b>slithey</b> toves did gyre and
```

CTX_DOC.SNIPPET returns one or more most relevant fragments for a document that contains the query term. Because CTX_DOC.SNIPPET returns surrounding text, you can immediately evaluate how useful the returned term is. CTX_DOC.SNIPPET returns the entire document if no words in the returned text are marked up.

> **See Also:** CTX_DOC.POLICY_SNIPPET on page 8-29 for a policy-based version of this procedure

Syntax

Syntax 1

```
exec CTX_DOC.SNIPPET(
index_name              IN VARCHAR2,
textkey                  IN VARCHAR2,
text_query               IN VARCHAR2,
starttag                 IN VARCHAR2 DEFAULT '<b>',
endtag                   IN VARCHAR2 DEFAULT '</b>',
entity_translation       IN BOOLEAN  DEFAULT TRUE,
separator                IN VARCHAR2 DEFAULT '<b>...</b>'
)
return varchar2;
```

Syntax 2

```
exec CTX_DOC.SNIPPET_CLOB_QUERY(
index_name               IN VARCHAR2,
textkey                  IN VARCHAR2,
text_query               IN CLOB,
starttag                 IN VARCHAR2 DEFAULT '<b>',
endtag                   IN VARCHAR2 DEFAULT '</b>',
entity_translation       IN BOOLEAN DEFAULT TRUE,
separator                IN VARCHAR2 DEFAULT '<b>...</b>'
)
return varchar2;
```

index_name

Specify the name of the index for the text column.

textkey

Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can be as follows:

- A single column primary key value

- An encoded specification for a composite (multiple column) primary key. When textkey is a composite key, you must encode the composite textkey string using the CTX_DOC.PKENCODE procedure.
- The rowid of the row containing the document

Use CTX_DOC.SET_KEY_TYPE to toggle between primary key and rowid identification.

text_query
Specify the original query expression used to retrieve the document. If NULL, no highlights are generated.

If text_query includes wildcards, stemming, fuzzy matching which result in stopwords being returned, SNIPPET does not highlight the stopwords.

If text_query contains the threshold operator, the operator is ignored.

starttag
Specify the start tag for marking up the query keywords. Default is '<*b*>'.

endtag
Specify the end tag for marking up the query keywords. Default is '</*b*>'.

entity_translation
Specify if you want HTML entities to be translated. The default is TRUE, which means that the special entities (<, >, and *&*) are translated into their alternative forms ('*<*', '*>*', and '*&*') when output by the procedure. However, special characters in the markup tags that are generated by CTX_DOC.SNIPPET will not be translated.

separator
Specify the string separating different returned fragments. Default is '<*b*>...</*b*>'.

Example

```
create table tdrbhk01 (id number primary key, text varchar2(4000));

insert into tdrbhk01 values (1, 'Oracle Text adds powerful search
<title>withintitle</title> and intelligent text management to the Oracle
database.  Complete.  You can search and manage documents, web pages,
catalog entries in more than 150 formats in any language.  Provides a
complete text query language and complete character support.  Simple.  You
can index and search text using SQL. Oracle Text Management can be done
using Oracle Enterprise Manager - a GUI tool.  Fast.  You can search
millions of documents, document,web pages, catalog entries using the
power and scalability of the database.  Intelligent.  Oracle Text''s
unique knowledge-base enables you to search, classify, manage
documents, clusters and summarize text based on its meaning as well as
its content. ');

exec ctx_ddl.create_section_group('my_sectioner','BASIC_SECTION_GROUP');
exec ctx_ddl.add_field_section('my_sectioner','title','title', false);

create index tdrbhk01x on tdrbhk01(text) indextype is ctxsys.context
  parameters ('filter CTXSYS.NULL_FILTER
               section group my_sectioner
               nopopulate');

select ctx_doc.snippet('tdrbhk01x','1',
              'search | classify') from dual;
```

The result looks something like this:

```
CTX_DOC.SNIPPET('TDRBHK01X','1','SEARCH|CLASSIFY')
--------------------------------------------------------------------------

Text's unique knowledge-base enables you to <b>search</b>,
<b>classify</b>, manage documents, clusters and summarize
```

Limitations

CTX_DOC.SNIPPET does not support the use of query templates.

CTX_DOC.SNIPPET displays marked-up keywords in context when used with NULL_SECTION_GROUP. However, there are limitations when using this procedure with XML documents. When used with XML_SECTION_GROUP or AUTO_SECTION_GROUP, the XML structure is ignored and user-specified tags are stripped out, which results in parts of surrounding text to be included in the returned snippet.

Related Topics

See Also: "POLICY_SNIPPET" on page 8-29, "HIGHLIGHT" on page 8-10, and "MARKUP" on page 8-14

THEMES

Use the CTX_DOC.THEMES procedure to generate a list of themes for a document. You can store each theme as a row in either a result table or an in-memory PL/SQL table that you specify.

> **Note:** CTX_DOC.THEMES requires an installed knowledge base. A knowledge base may or may not have been installed with Oracle Text. For more information on knowledge bases, see *Oracle Text Application Developer's Guide*.

Syntax 1: In-Memory Table Storage

```
CTX_DOC.THEMES(
index_name      IN VARCHAR2,
textkey         IN VARCHAR2,
restab          IN OUT NOCOPY THEME_TAB,
full_themes     IN BOOLEAN DEFAULT FALSE,
num_themes       IN NUMBER DEFAULT 50);
```

Syntax 2: Result Table Storage

```
CTX_DOC.THEMES(
index_name      IN VARCHAR2,
textkey         IN VARCHAR2,
restab          IN VARCHAR2,
query_id        IN NUMBER DEFAULT 0,
full_themes     IN BOOLEAN DEFAULT FALSE,
num_themes       IN NUMBER DEFAULT 50);
```

index_name

Specify the name of the index for the text column.

textkey

Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can be as follows:

- A single column primary key value
- An encoded specification for a composite (multiple column) primary key. When textkey is a composite key, you must encode the composite textkey string using the CTX_DOC.PKENCODE procedure.
- The rowid of the row containing the document

Toggle between primary key and rowid identification using CTX_DOC.SET_KEY_TYPE.

restab

You can specify this procedure to store results to either a table or to an in-memory PL/SQL table.

To store results in a table, specify the name of the table.

> **See Also:** "Theme Table" on page A-7 in Appendix A, "Oracle Text Result Tables"

To store results in an in-memory table, specify the name of the in-memory table of type THEME_TAB. The THEME_TAB datatype is defined as follows:

```
type theme_rec is record (
   theme varchar2(2000),
   weight number
);

type theme_tab is table of theme_rec index by binary_integer;
```

CTX_DOC.THEMES clears the THEME_TAB you specify before the operation.

query_id
Specify the identifier used to identify the row(s) inserted into restab.

full_themes
Specify whether this procedure generates a single theme or a hierarchical list of parent themes (full themes) for each document theme.

Specify TRUE for this procedure to write full themes to the THEME column of the result table.

Specify FALSE for this procedure to write single theme information to the THEME column of the result table. This is the default.

num_themes
Specify the maximum number of themes to retrieve. For example, if you specify 10, then up to the first 10 themes are returned for the document. The default is 50.

If you specify 0 or NULL, then this procedure returns all themes in a document. If the document contains more than 50 themes, then only the first 50 themes show conceptual hierarchy.

Examples

In-Memory Themes

The following example generates the first 10 themes for document 1 and stores them in an in-memory table called the_themes. The example then loops through the table to display the document themes.

```
declare
 the_themes ctx_doc.theme_tab;

begin
 ctx_doc.themes('myindex','1',the_themes, num_themes=>10);
 for i in 1..the_themes.count loop
  dbms_output.put_line(the_themes(i).theme||':'||the_themes(i).weight);
  end loop;
end;
```

Theme Table

The following example creates a theme table called CTX_THEMES:

```
create table CTX_THEMES (query_id number,
                         theme varchar2(2000),
                         weight number);
```

Single Themes

To obtain a list of up to the first 20 themes, where each element in the list is a single theme, enter a statement like the following example:

```
begin
 ctx_doc.themes('newsindex','34','CTX_THEMES',1,full_themes => FALSE,
 num_themes=> 20);
end;
```

Full Themes

To obtain a list of the top 20 themes, where each element in the list is a hierarchical list of parent themes, enter a statement like the following example:

```
begin
ctx_doc.themes('newsindex','34','CTX_THEMES',1,full_themes => TRUE, num_
themes=>20);
end;
```

TOKENS

Use this procedure to identify all text tokens in a document. The tokens returned are those tokens that are inserted into the index. This feature is useful for implementing document classification, routing, or clustering.

Stopwords are not returned. Section tags are not returned because they are not text tokens.

Syntax 1: In-Memory Table Storage

```
CTX_DOC.TOKENS(index_name      IN VARCHAR2,
               textkey         IN VARCHAR2,
               restab          IN OUT NOCOPY TOKEN_TAB);
```

Syntax 2: Result Table Storage

```
CTX_DOC.TOKENS(index_name      IN VARCHAR2,
               textkey         IN VARCHAR2,
               restab          IN VARCHAR2,
               query_id        IN NUMBER DEFAULT 0);
```

index_name

Specify the name of the index for the text column.

textkey

Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can be as follows:

- A single column primary key value
- Encoded specification for a composite (multiple column) primary key. To encode a composite textkey, use the CTX_DOC.PKENCODE procedure.
- The rowid of the row containing the document

Toggle between primary key and rowid identification using CTX_DOC.SET_KEY_TYPE.

restab

You can specify that this procedure store results to either a table or to an in-memory PL/SQL table.

The tokens returned are those tokens that are inserted into the index for the document (or row) named with textkey. Stop words are not returned. Section tags are not returned because they are not text tokens.

Specifying a Token Table

To store results to a table, specify the name of the table. Token tables can be named anything, but must include the columns shown in the following table, with names and datatypes as specified.

Table 8–1 Required Columns for Token Tables

Column Name	Type	Description
QUERY_ID	NUMBER	The identifier for the results generated by a particular call to CTX_DOC.TOKENS (only populated when table is used to store results from multiple TOKEN calls)
TOKEN	VARCHAR2(64)	The token string in the text.
OFFSET	NUMBER	The position of the token in the document, relative to the start of document which has a position of 1.
LENGTH	NUMBER	The character length of the token.

Specifying an In-Memory Table

To store results to an in-memory table, specify the name of the in-memory table of type TOKEN_TAB. The TOKEN_TAB datatype is defined as follows:

```
type token_rec is record (
token varchar2(64),
offset number,
length number
);

type token_tab is table of token_rec index by binary_integer;
```

CTX_DOC.TOKENS clears the TOKEN_TAB you specify before the operation.

query_id
Specify the identifier used to identify the row(s) inserted into restab.

Example

In-Memory Tokens

The following example generates the tokens for document 1 and stores them in an in-memory table, declared as the_tokens. The example then loops through the table to display the document tokens.

```
declare
 the_tokens ctx_doc.token_tab;

begin
 ctx_doc.tokens('myindex','1',the_tokens);
 for i in 1..the_tokens.count loop
  dbms_output.put_line(the_tokens(i).token);
  end loop;
end;
```

9

CTX_OUTPUT Package

This chapter provides reference information for using the CTX_OUTPUT PL/SQL package.

CTX_OUTPUT contains the following stored procedures:

Name	Description
ADD_EVENT	Adds an event to the index log.
ADD_TRACE	Enables tracing.
DISABLE_QUERY_STATS	Turns off the gathering of query stats for the index.
ENABLE_QUERY_STATS	Enables gathering of query stats for the index.
END_LOG	Halts logging of index and document services requests.
END_QUERY_LOG	Stops logging queries into a logfile.
GET_TRACE_VALUE	Returns the value of a trace.
LOG_TRACES	Prints traces to logfile.
LOGFILENAME	Returns the name of the current log file.
REMOVE_EVENT	Removes an event from the index log.
REMOVE_TRACE	Disables tracing.
RESET_TRACE	Clears a trace.
START_LOG	Starts logging index and document service requests.
START_QUERY_LOG	Creates a log file of queries.

ADD_EVENT

Use this procedure to add an event to the index log for more detailed log output or to enable error tracing for Oracle Text errors.

Syntax

```
CTX_OUTPUT.ADD_EVENT(event in NUMBER, errnum in NUMBER := null);
```

event
Specify the type of index event to log. You can add the following events:

- `CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID`, which logs the rowid of each row after it is indexed. This is useful for debugging a failed index operation.
- `CTX_OUTPUT.EVENT_OPT_PRINT_TOKEN`, which prints each token as it is being optimized.
- `CTX_OUTPUT.EVENT_INDEX_PRINT_TOKEN`, which prints the each token as it is being indexed.
- `CTX_OUTPUT.EVENT_DRG_DUMP_ERRORSTACK`, which prints the stack trace for the specified DRG error in the log. An error will be raised if `errnum` is not specified.

errnum
Specify the DRG error number for a `CTX_OUTPUT.EVENT_DRG_DUMP_ERRRORSTACK` event.

Example

```
begin
CTX_OUTPUT.ADD_EVENT(CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID);
end;
```

Related Topics

See Also: "REMOVE_EVENT" on page 9-12

ADD_TRACE

Use this procedure to enable a trace. If the trace has not been enabled, this call adds the trace to the list of active traces and resets its value to 0. If the trace has already been enabled, an error is raised.

Syntax

```
CTX_OUTPUT.ADD_TRACE(trace_id BINARY_INTEGER);
```

trace_id
Specify the ID of the trace to enable. See Table 9–1 for possible trace values.

Notes

Table 9–1 shows the available traces:

Table 9–1 Available Traces

Symbol	ID	Metric
TRACE_IDX_USER_DATASTORE	1	Time spent executing user datastore
TRACE_IDX_AUTO_FILTER	2	Time spent invoking the AUTO_FILTER filter. (Replaces the deprecated TRACE_IDX_INSO_FILTER trace)
TRACE_QRY_XX_TIME	3	Time spent executing the $X cursor
TRACE_QRY_XF_TIME	4	Time spent fetching from $X
TRACE_QRY_X_ROWS	5	Total number of rows whose token metadata was fetched from $X
TRACE_QRY_IF_TIME	6	Time spent fetching the LOB locator from $I
TRACE_QRY_IR_TIME	7	Time spent reading $I LOB information
TRACE_QRY_I_ROWS	8	Number of rows whose $I token_info was actually read
TRACE_QRY_I_SIZE	9	Number of bytes read from $I LOBs
TRACE_QRY_R_TIME	10	Time spent fetching and reading $R information
TRACE_QRY_CON_TIME	11	Time spent in CONTAINS processing (drexrcontains/drexrstart/drexrfetch)
TRACE_QRY_S_TIME	15	Time spent fetching and reading $S information

Tracing is independent of logging. Logging does not have to be on to start tracing, and vice-versa.

Traces are associated with a session—they can measure operations that take place within a single session, and conversely, cannot make measurements across sessions.

During parallel sync or optimize, the trace profile will be copied to the slave sessions if and only if tracing is currently enabled. Each slave will accumulate its own traces and implicitly write all trace values to the slave logfile before termination.

Related Topics

See Also: "REMOVE_TRACE" on page 9-13, "GET_TRACE_VALUE" on page 9-9, "LOG_TRACES" on page 9-10,and "RESET_TRACE" on page 9-14, as well as *Oracle Text Application Developer's Guide*

DISABLE_QUERY_STATS

Disables gathering of query stats for the index.

Syntax

```
ctx_output.disable_query_stats(
index_name IN VARCHAR2
);
```

index_name
The name of the index on which query stats collection is to be disabled.

Example

Turn off gathering of query stats for the index `myindex`.

```
CTX_OUTPUT.DISABLE_QUERY_STATS(myindex);
```

Notes

Once the query stats is disabled for an index, gathering and storing query-related metadata is stopped for that index. All the entries corresponding to that index are cleared from the dictionary tables. An error is returned if you call this procedure on an index where query stats is not enabled.

Related Topics

CTX_OUTPUT.ENABLE_QUERY_STATS on page 9-6, and CTX_REPORT.INDEX_STATS on page 11-8.

ENABLE_QUERY_STATS

Enables gathering of query stats for the index. To have query-related metadata stored for the index, use this procedure to enable collection of statistics on that index. You can only access the gathered metadata when `ctx_output.enable_query_stats` is turned on for the index.

> **Note:** Accessing the query stats metadata only works when `ctx_output.enable_query_stats` is turned on for the index. Please see CTX_REPORT.INDEX_STATS for the list of gathered query stats metadata.

Syntax

```
ctx_output.enable_query_stats(
index_name IN VARCHAR2
);
```

index_name
The name of the index on which query stats collection is to be enabled.

Example

Turn on gathering of query stats for the index `myindex`.

```
CTX_OUTPUT.ENABLE_QUERY_STATS(myindex);
```

Notes

The information that shows whether query stats is enabled on an index is available in the views: `CTX_INDEXES` and `CTX_USER_INDEXES` under the column `idx_query_stats_enabled`, which is in both of these views. If `query_stats` is enabled for an index, then the column displays YES; if not, then the column displays NO.

The data corresponding to the query statistics will be stored in persistent dictionary tables. Once `statistics` has been enabled for a particular index, query statistics will be collected for that index from all sessions.

If you call this procedure for an index where query stats is already enabled, then an error is thrown.

Statistics collection has a minimal effect on query performance.

Related Topics

CTX_OUTPUT.DISABLE_QUERY_STATS, and CTX_REPORT.INDEX_STATS.

END_LOG

This procedure halts logging index and document service requests.

Syntax

```
ctx_output.end_log;
```

Example

```
begin
CTX_OUTPUT.END_LOG;
end;
```

END_QUERY_LOG

Use this procedure to stop logging queries into a logfile created with CTX_OUTPUT.START_QUERY_LOG.

Syntax

```
ctx_output.end_query_log;
```

Example

```
begin
CTX_OUTPUT.START_QUERY_LOG('mylog1');
     < get queries >
CTX_OUTPUT.END_QUERY_LOG;
end;
```

GET_TRACE_VALUE

Use this procedure to programmatically retrieve the current value of a trace.

Syntax

```
CTX_OUTPUT.GET_TRACE_VALUE(trace_id BINARY_INTEGER);
```

trace_id
Specify the trace ID whose value you want. See Table 9–1, " Available Traces" on page 9-3 for possible values.

Example

This sets the value of the variable *value*:

```
value := ctx_output.get_trace_value(trace_id);
```

Notes

You can also retrieve trace values through SQL:

```
select * from ctx_trace_values;
```

See "CTX_TRACE_VALUES" on page G-11 for the entries in the `CTX_TRACE_VALUES` view.

If the trace has not been enabled, an error is raised.

Traces are not reset to 0 by this call.

Traces are associated with a session—they can measure operations that take place within a single session, and conversely, cannot make measurements across sessions.

Related Topics

See Also: "REMOVE_TRACE" on page 9-13, "ADD_TRACE" on page 9-3, "LOG_TRACES" on page 9-10,and "RESET_TRACE" on page 9-14, as well as the *Oracle Text Application Developer's Guide*

LOG_TRACES

Use this procedure to print all active traces to the logfile.

Syntax

```
CTX_OUTPUT.LOG_TRACES;
```

Notes

If logging has not been started, an error is raised.

Traces are not reset to 0 by this call.

This procedure looks for the logfile in the directory specified by the `LOG_DIRECTORY` system parameter, which is `$ORACLE_HOME/ctx/log` on UNIX. You can query the `CTX_PARAMETERS` view to find the current setting.

Related Topics

See Also: "REMOVE_TRACE" on page 9-13, "GET_TRACE_VALUE" on page 9-9, "ADD_TRACE" on page 9-3, and "RESET_TRACE" on page 9-14, as well as the *Oracle Text Application Developer's Guide*

LOGFILENAME

Returns the filename for the current log. This procedure looks for the logfile in the directory specified by the LOG_DIRECTORY system parameter, which is $ORACLE_HOME/ctx/log on UNIX. You can query the CTX_PARAMETERS view to find the current setting.

Syntax

```
CTX_OUTPUT.LOGFILENAME RETURN VARCHAR2;
```

Returns

Log file name.

Example

```
declare
   logname varchar2(100);
begin
   logname := CTX_OUTPUT.LOGFILENAME;
   dbms_output.put_line('The current log file is: '||logname);
end;
```

REMOVE_EVENT

Use this procedure to remove an event from the index log.

Syntax

```
CTX_OUTPUT.REMOVE_EVENT(event in NUMBER);
```

event
Specify the type of index event to remove from the log. You can remove the following events:

- `CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID`, which logs the rowid of each row after it is indexed. This is useful for debugging a failed index operation.
- `CTX_OUTPUT.EVENT_OPT_PRINT_TOKEN`, which prints each token as it is being optimized.
- `CTX_OUTPUT.EVENT_INDEX_PRINT_TOKEN`, which prints the each token as it is being indexed.

Example

```
begin
CTX_OUTPUT.REMOVE_EVENT(CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID);
end;
```

Related Topics

See Also: "ADD_EVENT" on page 9-2

REMOVE_TRACE

Use this procedure to disable a trace.

Syntax

```
CTX_OUTPUT.REMOVE_TRACE(trace_id BINARY_INTEGER);
```

trace_id
Specify the ID of the trace to disable. See Table 9–1, " Available Traces" on page 9-3 for possible values.

Notes

If the trace has not been enabled, an error is raised.

Related Topics

See Also: "GET_TRACE_VALUE" on page 9-9, "ADD_TRACE" on page 9-3, "LOG_TRACES" on page 9-10,and "RESET_TRACE" on page 9-14, as well as the *Oracle Text Application Developer's Guide*

RESET_TRACE

Use this procedure to clear a trace (that is, reset it to 0).

Syntax

```
CTX_OUTPUT.RESET_TRACE(trace_id BINARY_INTEGER);
```

trace_id
Specify the ID of the trace to reset. See Table 9–1, " Available Traces" on page 9-3 for possible values.

Notes

If the trace has not been enabled, an error is raised.

Related Topics

See Also: "REMOVE_TRACE" on page 9-13, "GET_TRACE_VALUE" on page 9-9, "ADD_TRACE" on page 9-3, "LOG_TRACES" on page 9-10, as well as the *Oracle Text Application Developer's Guide*

START_LOG

Begin logging index and document service requests.

Syntax

```
CTX_OUTPUT.START_LOG(logfile in varchar2, overwrite in default true);
```

logfile
Specify the name of the log file. The log is stored in the directory specified by the system parameter LOG_DIRECTORY.

overwrite
Specify whether you want to overwrite or append to the original query log file specified by *logfile*, if it already exists. The default is to overwrite the original query log file.

Example

```
begin
CTX_OUTPUT.START_LOG('mylog1');
end;
```

Notes

Logging is independent of tracing. Logging does not have to be on to start tracing, and vice-versa.

Logging is associated with a session—it can log operations that take place within a single session, and, conversely, cannot make measurements across sessions.

Filenames used in CTX_OUTPUT.START_LOG are restricted to the following characters: alphanumeric, minus, period, space, hash, underscore, single and double quotes. Any other character in the filename will raise an error.

START_QUERY_LOG

Begin logging query requests into a query log file.

Use CTX_OUTPUT.END_QUERY_LOG to stop logging queries. Use CTX_REPORT.QUERY_LOG_SUMMARY to obtain reports on logged queries, such as which queries returned successfully the most times.

The query log includes the query string, the index name, and the timestamp of the query, as well as whether or not the query successfully returned a hit. A successful query for the phrase *Blues Guitarists* made at 6:46 (local time) on November 11th, 2003, would be entered into the query log in this form:

```
<QuerySet><TimeStamp>18:46:51 02/04/03</TimeStamp><IndexName>
IDX_SEARCH_TABLE</IndexName><Query>Blues
Guitarists</Query><ReturnHit>Yes</ReturnHit></QuerySet>
```

Syntax

```
CTX_OUTPUT.START_QUERY_LOG(logfile in varchar2, overwrite in default true);
```

logfile
Specify the name of the query log file. The query log is stored in the directory specified by the system parameter LOG_DIRECTORY.

overwrite
Specify whether you want to overwrite or append to the original query log file specified by *logfile*, if it already exists. The default is to overwrite the original query log file.

Example

```
begin
CTX_OUTPUT.START_QUERY_LOG('mylog1');
     < get queries >
CTX_OUTPUT.END_QUERY_LOG;
end;
```

Notes

Filenames used in CTX_OUTPUT.START_QUERY_LOG are restricted to the following characters: alphanumeric, minus, period, space, hash, underscore, single and double quotes. Any other character in the filename will raise an error.

Logging is associated with a session—it can log operations that take place within a single session, and, conversely, cannot make measurements across sessions.

10

CTX_QUERY Package

This chapter describes the CTX_QUERY PL/SQL package you can use for generating query feedback, counting hits, and creating stored query expressions.

> **Note:** You can use this package only when your index type is CONTEXT. This package does not support the CTXCAT index type.

The CTX_QUERY package includes the following procedures and functions:

Name	Description
BROWSE_WORDS	Returns the words around a seed word in the index.
COUNT_HITS	Returns the number hits to a query.
EXPLAIN	Generates query expression parse and expansion information.
HFEEDBACK	Generates hierarchical query feedback information (broader term, narrower term, and related term).
REMOVE_SQE	Removes a specified stored query expression from the SQL tables.
RESULT_SET	Executes a query and generates a result set.
STORE_SQE	Executes a query and stores the results in stored query expression tables.

BROWSE_WORDS

This procedure enables you to browse words in an Oracle Text index. Specify a seed word and BROWSE_WORDS returns the words around it in the index, and an approximate count of the number of documents that contain each word.

This feature is useful for refining queries. You can identify the following words:

- Unselective words (words that have low document count)
- Misspelled words in the document set

Syntax 1: To Store Results in Table

```
ctx_query.browse_words(
index_name  IN   VARCHAR2,
seed        IN   VARCHAR2,
restab      IN   VARCHAR2,
browse_id   IN   NUMBER   DEFAULT 0,
numwords    IN   NUMBER   DEFAULT 10,
direction   IN   VARCHAR2 DEFAULT BROWSE_AROUND,
part_name   IN   VARCHAR2 DEFAULT NULL
);
```

Syntax 2: To Store Results in Memory

```
ctx_query.browse_words(
index_name  IN      VARCHAR2,
seed        IN      VARCHAR2,
resarr      IN OUT  BROWSE_TAB,
numwords    IN      NUMBER   DEFAULT 10,
direction   IN      VARCHAR2 DEFAULT BROWSE_AROUND,
part_name   IN      VARCHAR2 DEFAULT NULL
);
```

index

Specify the name of the index. You can specify schema.name. Must be a local index.

seed

Specify the seed word. This word is lexed before browse expansion. The word need not exist in the token table. seed must be a single word. Using multiple words as the seed will result in an error.

restab

Specify the name of the result table. You can enter restab as schema.name. The table must exist before you call this procedure, and you must have INSERT permissions on the table. This table must have the following schema.

Column	Datatype
browse_id	number
word	varchar2(64)
doc_count	number

Existing rows in restab are not deleted before BROWSE_WORDS is called.

resarr
Specify the name of the result array. `resarr` is of type `ctx_query.browse_tab`.

```
type browse_rec is record (
   word varchar2(64),
   doc_count number
);
type browse_tab is table of browse_rec index by binary_integer;
```

browse_id
Specify a numeric identifier between 0 and 2^{32}. The rows produced for this browse have a value of in the `browse_id` column in `restab`. When you do not specify `browse_id`, the default is 0.

numwords
Specify the number of words returned.

direction
Specify the direction for the browse. You can specify one of:

value	behavior
`BEFORE`	Browse seed word and words alphabetically before the seed.
`AROUND`	Browse seed word and words alphabetically before and after the seed.
`AFTER`	Browse seed word and words alphabetically after the seed.

Symbols `CTX_QUERY.BROWSE_BEFORE`, `CTX_QUERY.BROWSE_AROUND`, and `CTX_QUERY.BROWSE_AFTER` are defined for these literal values as well.

part_name
Specify the name of the index partition to browse.

Example

Browsing Words with Result Table

```
begin
ctx_query.browse_words('myindex','dog','myres',numwords=>5,direction=>'AROUND');
end;

select word, doc_count from myres order by word;

WORD       DOC_COUNT
--------   ----------
CZAR       15
DARLING    5
DOC        73
DUNK       100
EAR        3
```

Browsing Words with Result Array

```
set serveroutput on;
declare
  resarr ctx_query.browse_tab;
begin
ctx_query.browse_words('myindex','dog',resarr,5,CTX_QUERY.BROWSE_AROUND);
for i in 1..resarr.count loop
```

```
  dbms_output.put_line(resarr(i).word || ':' || resarr(i).doc_count);
end loop;
end;
```

COUNT_HITS

Returns the number of hits for the specified query. You can call COUNT_HITS in exact or estimate mode. Exact mode returns the exact number of hits for the query. Estimate mode returns an upper-bound estimate but runs faster than exact mode.

Syntax

Syntax 1

```
exec CTX_QUERY.COUNT_HITS(
    index_name  IN VARCHAR2,
    text_query  IN VARCHAR2,
    exact       IN BOOLEAN  DEFAULT TRUE,
    part_name   IN VARCHAR2 DEFAULT NULL
) RETURN NUMBER;
```

Syntax 2

```
exec CTX_QUERY.COUNT_HITS_CLOB_QUERY(
    index_name  IN VARCHAR2,
    text_query  IN CLOB,
    exact       IN BOOLEAN DEFAULT TRUE,
    part_name   IN VARCHAR2 DEFAULT NULL
) RETURN NUMBER;
```

index_name
Specify the index name.

text_query
Specify the query.

exact
Specify TRUE for an exact count. Specify FALSE for an upper-bound estimate.

Specifying FALSE returns a less accurate number but runs faster. Specifying FALSE might return a number which is too high if rows have been updated or deleted since the last FULL index optimize. Optimizing in full mode removes these false hits, and then EXACT set to FALSE will return the same number as EXACT set to TRUE.

part_name
Specify the name of the index partition to query.

Notes

If the query contains structured criteria, then you should use SELECT COUNT(*).

If the index was created with the TRANSACTIONAL parameter, then COUNT_HITS will include pending rowids as well as those that have been synchronized.

EXPLAIN

Use CTX_QUERY.EXPLAIN to generate explain plan information for a query expression. The EXPLAIN plan provides a graphical representation of the parse tree for a Text query expression. This information is stored in a result table.

This procedure does *not* execute the query. Instead, this procedure can tell you how a query is expanded and parsed before you enter the query. This is especially useful for stem, wildcard, thesaurus, fuzzy, soundex, or about queries. Parse trees also show the following information:

- Order of execution (precedence of operators)
- ABOUT query normalization
- Query expression optimization
- Stop-word transformations
- Breakdown of composite-word tokens

Knowing how Oracle Text evaluates a query is useful for refining and debugging queries. You can also design your application so that it uses the explain plan information to help users write better queries.

Syntax

Syntax 1

```
exec CTX_QUERY.EXPLAIN(
    index_name     IN VARCHAR2,
    text_query     IN VARCHAR2,
    explain_table  IN VARCHAR2,
    sharelevel     IN NUMBER DEFAULT 0,
    explain_id     IN VARCHAR2 DEFAULT NULL,
    part_name      IN VARCHAR2 DEFAULT NULL
);
```

Syntax 2

```
exec CTX_QUERY.EXPLAIN_CLOB_QUERY(
    index_name     IN VARCHAR2,
    text_query     IN CLOB,
    explain_table  IN VARCHAR2,
    sharelevel     IN NUMBER DEFAULT 0,
    explain_id     IN VARCHAR2 DEFAULT NULL,
    part_name      IN VARCHAR2 DEFAULT NULL
);
```

index_name

Specify the name of the index to be queried.

text_query

Specify the query expression to be used as criteria for selecting rows.

When you include a wildcard, fuzzy, or soundex operator in text_query, this procedure looks at the index tables to determine the expansion.

Wildcard, fuzzy (?), and soundex (!) expression feedback does not account for lazy deletes as in regular queries.

explain_table
Specify the name of the table used to store representation of the parse tree for *text_query*. You must have at least `INSERT` and `DELETE` privileges on the table used to store the results from `EXPLAIN`.

> **See Also:** "EXPLAIN Table" in Appendix A, "Oracle Text Result Tables" for more information about the structure of the explain table.

sharelevel
Specify whether `explain_table` is shared by multiple `EXPLAIN` calls. Specify 0 for exclusive use and 1 for shared use. Default is 0 (single-use).

When you specify 0, the system automatically truncates the result table before the next call to `EXPLAIN`.

When you specify 1 for shared use, this procedure does not truncate the result table. Only results with the same `explain_id` are updated. When no results with the same `explain_id` exist, new results are added to the `EXPLAIN` table.

explain_id
Specify a name that identifies the explain results returned by an `EXPLAIN` procedure when more than one `EXPLAIN` call uses the same shared `EXPLAIN` table. Default is `NULL`.

part_name
Specify the name of the index partition to query.

Example

Creating the Explain Table

To create an explain table called `test_explain` for example, use the following SQL statement:

```
create table test_explain(
         explain_id varchar2(30),
         id number,
         parent_id number,
         operation varchar2(30),
         options varchar2(30),
         object_name varchar2(64),
         position number,
         cardinality number);
```

Running CTX_QUERY.EXPLAIN

To obtain the expansion of a query expression such as *comp% OR ?smith*, use `CTX_QUERY.EXPLAIN` as follows:

```
ctx_query.explain(
         index_name => 'newindex',
         text_query => 'comp% OR ?smith',
         explain_table => 'test_explain',
         sharelevel => 0,
         explain_id => 'Test');
```

Retrieving Data from Explain Table

To read the explain table, you can select the columns as follows:

```
select explain_id, id, parent_id, operation, options, object_name, position
from test_explain order by id;
```

The output is ordered by ID to simulate a hierarchical query:

```
EXPLAIN_ID    ID PARENT_ID OPERATION    OPTIONS OBJECT_NAME POSITION
---------- ---- --------- ------------ ------- ----------- --------
Test           1         0 OR           NULL    NULL          1
Test           2         1 EQUIVALENCE  NULL    COMP%         1
Test           3         2 WORD         NULL    COMPTROLLER   1
Test           4         2 WORD         NULL    COMPUTER      2
Test           5         1 EQUIVALENCE  (?)     SMITH         2
Test           6         5 WORD         NULL    SMITH         1
Test           7         5 WORD         NULL    SMYTHE        2
```

Restrictions

`CTX_QUERY.EXPLAIN` does not support the use of query templates.

You cannot use `CTX_QUERY.EXPLAIN` with remote queries.

If the query utilizes themes (for example, with an `ABOUT` query), then a knowledge base must be installed. Such a knowledge base may or may not have been installed with Oracle Text. For more information on knowledge bases, see *Oracle Text Application Developer's Guide*.

Related Topics

Chapter 3, "Oracle Text CONTAINS Query Operators"

Appendix H, "Stopword Transformations in Oracle Text"

HFEEDBACK

In English or French, this procedure generates hierarchical query feedback information (broader term, narrower term, and related term) for the specified query.

Broader term, narrower term, and related term information is obtained from the knowledge base. However, only knowledge base terms that are also in the index are returned as query feedback information. This increases the chances that terms returned from HFEEDBACK produce hits over the currently indexed document set.

Hierarchical query feedback information is useful for suggesting other query terms to the user.

Note: CTX_QUERY.HFEEDBACK requires an installed knowledge base. A knowledge base may or may not have been installed with Oracle Text. For more information on knowledge bases, see *Oracle Text Application Developer's Guide*.

CTX_QUERY.HFEEDBACK is only supported in English and French.

Syntax

Syntax 1

```
exec CTX_QUERY.HFEEDBACK(
           index_name     IN VARCHAR2,
           text_query     IN VARCHAR2,
           feedback_table IN VARCHAR2,
           sharelevel     IN NUMBER DEFAULT 0,
           feedback_id    IN VARCHAR2 DEFAULT NULL,
           part_name      IN VARCHAR2 DEFAULT NULL
);
```

Syntax 2

```
exec CTX_QUERY.HFEEDBACK_CLOB_QUERY(
           index_name     IN VARCHAR2,
           text_query     IN CLOB,
           feedback_table IN VARCHAR2,
           sharelevel     IN NUMBER DEFAULT 0,
           feedback_id    IN VARCHAR2 DEFAULT NULL,
           part_name      IN VARCHAR2 DEFAULT NULL
);
```

index_name
Specify the name of the index for the text column to be queried.

text_query
Specify the query expression to be used as criteria for selecting rows.

feedback_table
Specify the name of the table used to store the feedback terms.

> **See Also:** "HFEEDBACK Table" in Appendix A, "Oracle Text Result Tables" for more information about the structure of the explain table.

sharelevel
Specify whether feedback_table is shared by multiple HFEEDBACK calls. Specify 0 for exclusive use and 1 for shared use. Default is 0 (single-use).

When you specify 0, the system automatically truncates the feedback table before the next call to HFEEDBACK.

When you specify 1 for shared use, this procedure does not truncate the feedback table. Only results with the same feedback_id are updated. When no results with the same feedback_id exist, new results are added to the feedback table.

feedback_id
Specify a value that identifies the feedback results returned by a call to HFEEDBACK when more than one HFEEDBACK call uses the same shared feedback table. Default is NULL.

part_name
Specify the name of the index partition to query.

Example

Create HFEEDBACK Result Table

Create a result table to use with CTX_QUERY.HFEEDBACK as follows:

```
CREATE TABLE restab (
  feedback_id VARCHAR2(30),
  id          NUMBER,
  parent_id   NUMBER,
  operation   VARCHAR2(30),
  options     VARCHAR2(30),
  object_name VARCHAR2(80),
  position    NUMBER,
  bt_feedback ctxsys.ctx_feedback_type,
  rt_feedback ctxsys.ctx_feedback_type,
  nt_feedback ctxsys.ctx_feedback_type
) NESTED TABLE bt_feedback STORE AS res_bt
  NESTED TABLE rt_feedback STORE AS res_rt
  NESTED TABLE nt_feedback STORE AS res_nt;
```

CTX_FEEDBACK_TYPE is a system-defined type in the CTXSYS schema.

> **See Also:** "HFEEDBACK Table" in Appendix A, "Oracle Text Result Tables" for more information about the structure of the HFEEDBACK table.

Call CTX_QUERY.HFEEDBACK

The following code calls the HFEEDBACK procedure with the query *computer industry*.

```
BEGIN
ctx_query.hfeedback (index_name     => 'my_index',
                     text_query     => 'computer industry',
                     feedback_table => 'restab',
                     sharelevel     => 0,
                     feedback_id    => 'query10'
                    );
END;
```

Select From the Result Table

The following code extracts the feedback data from the result table. It extracts broader term, narrower term, and related term feedback separately from the nested tables.

```
DECLARE
  i NUMBER;
BEGIN
  FOR frec IN (
    SELECT object_name, bt_feedback, rt_feedback, nt_feedback
    FROM restab
    WHERE feedback_id = 'query10' AND object_name IS NOT NULL
  ) LOOP

    dbms_output.put_line('Broader term feedback for ' || frec.object_name ||
':');
    i := frec.bt_feedback.FIRST;
    WHILE i IS NOT NULL LOOP
      dbms_output.put_line(frec.bt_feedback(i).text);
      i := frec.bt_feedback.NEXT(i);
    END LOOP;

    dbms_output.put_line('Related term feedback for ' || frec.object_name ||
':');
    i := frec.rt_feedback.FIRST;
    WHILE i IS NOT NULL LOOP
      dbms_output.put_line(frec.rt_feedback(i).text);
      i := frec.rt_feedback.NEXT(i);
    END LOOP;

    dbms_output.put_line('Narrower term feedback for ' || frec.object_name ||
':');
    i := frec.nt_feedback.FIRST;
    WHILE i IS NOT NULL LOOP
      dbms_output.put_line(frec.nt_feedback(i).text);
      i := frec.nt_feedback.NEXT(i);
    END LOOP;

  END LOOP;
END;
```

Sample Output

The following output is for the preceding example, which queries on *computer industry*:

```
Broader term feedback for computer industry:
hard sciences
Related term feedback for computer industry:
computer networking
electronics
knowledge
library science
mathematics
optical technology
robotics
satellite technology
semiconductors and superconductors
symbolic logic
telecommunications industry
Narrower term feedback for computer industry:
ABEND - abnormal end of task
AT&T Starlans
```

```
ATI Technologies, Incorporated
ActivCard
Actrade International Ltd.
Alta Technology
Amiga Format
Amiga Library Services
Amiga Shopper
Amstrat Action
Apple Computer, Incorporated
..
```

> **Note:** The HFEEDBACK information you obtain depends on the contents of your index and knowledge base and as such might differ from the sample shown.

Restrictions

CTX_QUERY.HFEEDBACK does not support the use of query templates.

REMOVE_SQE

The CTX_QUERY.REMOVE_SQE procedure removes the specified stored query expression.

Syntax

```
CTX_QUERY.REMOVE_SQE(query_name IN VARCHAR2);
```

query_name
Specify the name of the stored query expression to be removed.

Example

```
begin
ctx_query.remove_sqe('disasters');
end;
```

RESULT_SET

This procedure executes an XML query and generates a result set in XML. The result set interface can return data views that are difficult to express in SQL, such as top N by category queries.

> **See Also:** *Oracle Text Application Developer's Guide* for details on how to use the result set interface

Syntax

```
CTX_QUERY.RESULT_SET (
   index_name            IN VARCHAR2,
   query                 IN VARCHAR2,
   result_set_descriptor IN CLOB,
   result_set            IN OUT NOCOPY CLOB,
   part_name             IN VARCHAR2 DEFAULT NULL);
```

index_name
Specify the index against which to execute the query.

query
Specify the query string.

result_set_descriptor
Specify the result set descriptor in XML. It describes what the result set should contain. See "The Input Result Set Descriptor" on page 10-14 for more details.

result_set
Specify the output result set. If this variable is `NULL` on input, a session-duration temporary lob will be allocated and returned to the user. The user is responsible for deallocating this temporary lob. See "The Output Result Set XML" on page 10-16 for more details.

part_name
Specify the index partition name. If the index is global, `part_name` must be `NULL`. If the index is partitioned and `part_name` is not `NULL`, then the query will only be evaluated for the given partition. If the index is partitioned and `part_name` is `NULL`, then the query will be evaluated for all partitions.

The Input Result Set Descriptor

The result set descriptor is an XML message which describes what to calculate for the result set. The elements present in the result set descriptor and the order in which they occur serve as a simple template, specifying what to include in the output result set. That is, there should be the list of hit rowids, then a count, then a token count, and so on. The attributes of the elements specify the parameters and options to the specific operations, such as number of hits in the list of rowids, estimate versus exact count, and so on.

The result set descriptor itself is XML conforming to the following DTD:

```
<!ELEMENT ctx_result_set_descriptor (hitlist?, group*, count?)>
<!ELEMENT hitlist (rowid?, score?, sdata*)>
<!ELEMENT group(count?)>
<!ELEMENT count EMPTY>
<!ELEMENT rowid EMPTY>
```

```
<!ELEMENT score EMPTY>
<!ELEMENT sdata EMPTY>
<!ATTLIST group sdata CDATA #REQUIRED>
<!ATTLIST hitlist start_num_hit integer #REQUIRED>
<!ATTLIST hitlist end_num_hit integer #REQUIRED>
<!ATTLIST hitlist order PCDATA #IMPLIED>
<!ATTLIST count exact (TRUE|FALSE) "FALSE">
<!ATTLIST sdata name CDATA #REQUIRED>
```

The following is a description of the possible XML elements for the result set descriptor:

- `ctx_result_set_descriptor`

  This is the root element for the result set descriptor. The parent element is none, as are the available attributes.

  The possible child elements are:

  – Zero or more `hitlist` elements.

  – Zero or more `group` elements.

  – At most one `count` element.

- `group`

  The `group` element causes the generated result set to include a group breakdown. In other words, a breakdown of the results by `SDATA` section values. The parent element is `ctx_result_set_descriptor`, and the available attributes are:

  – `sdata`

    Specifies the name of the `SDATA` section to use for grouping. It is required.

  Possible child elements of `group` are:

  – At most one `count` element.

- `hitlist`

  The `hitlist` element controls inclusion of a list of hit documents. The parent element is `ctx_result_set_descriptor`, and the available attributes are:

  – `start_hit_num`

    This specifies the starting document hit to be included in the generated result set. This can be set to any positive integer less than or equal to 2048. For example, if `start_hit_num` is 21, then the result set will include document hits starting from the 21st document hit. This element is required.

  – `end_hit_num`

    This specifies the last document hit to be included in the generated result set. This can be set to any positive integer less than or equal to 2048. For example, if `end_hit_num` is 40, then the result set will include document hits up to the 40th document hit. This element is required.

  – `order`

    This is an optional attribute that specifies the order for the documents in the generated result set. The value is a list similar to a SQL `ORDER BY` statement, except that, instead of column names, they can either be `SCORE` or `SDATA` section names. In the following example, `MYDATE` and `MYPRICE` are the `SDATA` section names:

    ```
    (order = "SCORE DESC, MYDATE, MYPRICE DESC")
    ```

The possible child elements are:

- At most one `rowid` element.
- At most one `score` element.
- At most one `sdata` element.

- `count`

  This element causes the generated result set to include a count of the number of hit documents. The parent elements are:

  - `ctx_result_set_descriptor`
  - `group`

  The available attributes are:

  - `exact`

    This is to estimate mode. Set to `true` or `false`. It is required, and the default is `false`.

  The possible child elements are none.

- `rowid`

  This child element causes the generated result set to include rowid information for each hit. The parent element is `hitlist`. There are no attributes and no possible child elements.

- `score`

  This child element causes the generated result set to include score information for each hit.

  - The parent element is `hitlist`.
  - There are no available attributes, and no possible child elements.

- `sdata`

  This child element causes the generated result set to include `sdata` values for each hit.

  - The parent element is `hitlist`.
  - The available attribute is `name`. This specifies the name of the `sdata` section. It is required.
  - There are no child elements.

The Output Result Set XML

The output result set XML is XML conforming to the following DTD:

```
<!ELEMENT ctx_result_set (hitlist?, groups*, count?)>
<!ELEMENT hitlist (hit*)>
<!ELEMENT hit(rowid?, score?, sdata*)>
<!ELEMENT groups (group*)>
<!ELEMENT group (count?)>
<!ELEMENT count CDATA>
<!ELEMENT rowid CDATA>
<!ELEMENT score CDATA>
<!ELEMENT sdata CDATA>
<!ATTLIST groups sdata CDATA #REQUIRED>
```

```
<!ATTLIST group value CDATA #REQUIRED>
<!ATTLIST sdata name CDATA #REQUIRED>
```

The following is a description of the list of possible XML elements for the output result set:

- `ctx_result_set`

  This is the root element for the generated result set. There are no attributes. The parent is none. The possible child elements are:

  - At most one `hitlist` element.
  - Zero or more `groups` elements.

- `groups`

  This delimits the start of a group breakdown section. The parent element is `ctx_result_set`. The available attributes are:

  - `sdata`

    This is the name of the `sdata` section used for grouping.

  The possible child elements are:

  - Zero or more `group` elements.

- `group`

  This delimits the start of a `GROUP BY` value. The parent element is the `groups` element. The available attributes are:

  - `value`

    This is the value of the `sdata` section.

  The possible child elements are at most one `count` element.

- `hitlist`

  This delimits the start of `hitlist` information. The parent element is `ctx_result_set`, while the children are zero or more `hit` elements. There are no attributes.

- `hit`

  This delimits the start of the information for a particular document within a `hitlist`. The parent element is `hitlist`, and there are no available attributes. The possible child elements are:

  - Zero or one `rowid` elements.
  - Zero or one `score` element.
  - Zero or one `sdata` element.

- `rowid`

  This is the rowid of the document, so the content is the rowid of the document. The parent element is the `hit` element. There are no child elements, and no available attributes.

- `score`

  This is the score of the document. The parent element is the hit element. The content is the numeric score. There are no available attributes, and no possible child elements.

- sdata

  This is the SDATA value or values for the document. The parent element is the hit element, and the available attribute is name, which is the name of the sdata section. There are no possible child elements available. The content is the SDATA section value, which, for DATE values, is in the format "YYYY-MM-DD HH24:MI:SS", depending upon the actual values being stored.

- count

  This is the document hit count. The parent element is the ctx_result_set element or the group element. It contains the numeric hit count, and has no attributes, and no possible child elements.

Example

This call to ctx_query.result_set() with the specified XML result_set_descriptor will generate the following information in the form of XML:

- top 20 hits displaying, score, rowid, author SDATA section value, and pubDate SDATA section value, order by pubDate SDATA section value DESC and score DESC
- total doc hit count for the text query
- counts group by pubDate SDATA section values
- counts group by author SDATA section values

```
declare
  rs clob;
begin
  dbms_lob.createtemporary(rs, true, dbms_lob.session);
  ctx_query.result_set('docidx', 'oracle', '
  <ctx_result_set_descriptor>
   <count/>
   <hitlist start_hit_num="1" end_hit_num="5" order="pubDate desc, score desc">
     <score/>
     <rowid/>
     <sdata name="author"/>
     <sdata name="pubDate"/>
   </hitlist>
   <group sdata="pubDate">
     <count/>
   </group>
   <group sdata="author">
     <count/>
   </group>
  </ctx_result_set_descriptor>
', rs);
  dbms_lob.freetemporary(rs);
exception
  when others then
   dbms_lob.freetemporary(rs);
   raise;
end;
/
```

The XML output store in the result set output clob will resemble the following:

```
<ctx_result_set>
  <hitlist>
    <hit>
      <score>3</score><rowid>AAAPoEAABAAAMWsAAC</rowid>
```

```
      <sdata name="AUTHOR">John</sdata>
      <sdata name="PUBDATE">2001-01-03 00:00:00</sdata>
    </hit>
    <hit>
      <score>3</score><rowid>AAAPoEAABAAAMWsAAG</rowid>
      <sdata name="AUTHOR">John</sdata>
      <sdata name="PUBDATE">2001-01-03 00:00:00</sdata>
    </hit>
    <hit>
      <score>3</score><rowid>AAAPoEAABAAAMWsAAK</rowid>
      <sdata name="AUTHOR">John</sdata>
      <sdata name="PUBDATE">2001-01-03 00:00:00</sdata>
    </hit>
    <hit>
      <score>3</score><rowid>AAAPoEAABAAAMWsAAO</rowid>
      <sdata name="AUTHOR">John</sdata>
      <sdata name="PUBDATE">2001-01-03 00:00:00</sdata>
    </hit>
    <hit>
      <score>3</score><rowid>AAAPoEAABAAAMWsAAS</rowid>
      <sdata name="AUTHOR">John</sdata>
      <sdata name="PUBDATE">2001-01-03 00:00:00</sdata>
    </hit>
  </hitlist>

  <count>100</count>

  <groups sdata="PUBDATE">
    <group value="2001-01-01 00:00:00"><count>25</count></group>
    <group value="2001-01-02 00:00:00"><count>50</count></group>
    <group value="2001-01-03 00:00:00"><count>25</count></group>
  </groups>

  <groups sdata="AUTHOR">
    <group value="John"><count>50</count></group>
    <group value="Mike"><count>25</count></group>
    <group value="Steve"><count>25</count></group>
  </groups>

</ctx_result_set>
```

STORE_SQE

This procedure creates a stored query expression. Only the query definition is stored.

Supported Operators

Stored query expressions support all of the CONTAINS query operators. Stored query expressions also support all of the special characters and other components that can be used in a query expression, including other stored query expressions.

Privileges

Users are allowed to create and remove stored query expressions owned by them. Users are allowed to use stored query expressions owned by anyone. The CTXSYS user can create or remove stored query expressions for any user.

Syntax

Syntax 1

```
CTX_QUERY.STORE_SQE(query_name      IN VARCHAR2,
                    text_query      IN VARCHAR2);
```

Syntax 2

```
CTX_QUERY.STORE_SQE_CLOB_QUERY(query_name   IN VARCHAR2,
                               text_query   IN CLOB);
```

query_name
Specify the name of the stored query expression to be created.

text_query
Specify the query expression to be associated with query_name.

Example

```
begin
ctx_query.store_sqe('disasters', 'hurricanes | earthquakes');
end;
```

11

CTX_REPORT Package

This chapter describes how to use the CTX_REPORT package to create reports on indexing and querying. These reports can help you troubleshoot problems or fine-tune your applications.

This chapter contains the following topics:

- Procedures in CTX_REPORT
- Using the Function Versions

> **See Also:** *Oracle Text Application Developer's Guide* for an overview of the CTX_REPORT package and how you can use the various procedures described in this chapter

Procedures in CTX_REPORT

The CTX_REPORT package contains the following procedures:

Name	Description
DESCRIBE_INDEX	Creates a report describing the index.
DESCRIBE_POLICY	Creates a report describing a policy.
CREATE_INDEX_SCRIPT	Creates a SQL*Plus script to duplicate the named index.
CREATE_POLICY_SCRIPT	Creates a SQL*Plus script to duplicate the named policy.
INDEX_SIZE	Creates a report to show the internal objects of an index, their tablespaces and used sizes.
INDEX_STATS	Creates a report to show the various statistics of an index.
QUERY_LOG_SUMMARY	Creates a report showing query statistics
TOKEN_INFO	Creates a report showing the information for a token, decoded.
TOKEN_TYPE	Translates a name and returns a numeric token type.

Using the Function Versions

Some of the procedures in the CTX_REPORT package have function versions. You can call these functions as follows:

```
select ctx_report.describe_index('MYINDEX') from dual;
```

In SQL*Plus, to generate an output file to send to support, you can do:

```
set long 64000
set pages 0
set heading off
set feedback off
spool outputfile
select ctx_report.describe_index('MYINDEX') from dual;
spool off
```

DESCRIBE_INDEX

Creates a report describing the index. This includes the settings of the index metadata, the indexing objects used, the settings of the attributes of the objects, and index partition descriptions, if any.

You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

Syntax

```
procedure CTX_REPORT.DESCRIBE_INDEX(
  index_name     IN VARCHAR2,
  report         IN OUT NOCOPY CLOB,
  report_format  IN VARCHAR2 DEFAULT FMT_TEXT
);

function CTX_REPORT.DESCRIBE_INDEX(
  index_name     IN VARCHAR2,
  report_format  IN VARCHAR2 DEFAULT FMT_TEXT
) return CLOB;
```

index_name
Specify the name of the index to describe.

report
Specify the CLOB locator to which to write the report.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

report_format
Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX_REPORT.FMT_TEXT or CTX_REPORT.FMT_XML.

Notes

CTX_REPORT.DESCRIBE_INDEX outputs FILTER BY and ORDER BY column information if the index is created with FILTER BY and/or ORDER BY clauses.

Related Topics

"CREATE INDEX" on page 1-36, and "ADD_SDATA_COLUMN" on page 7-14

DESCRIBE_POLICY

Creates a report describing the policy. This includes the settings of the policy metadata, the indexing objects used, the settings of the attributes of the objects.

You can call this operation as a procedure with an `IN OUT CLOB` parameter or as a function that returns the report as a `CLOB`.

Syntax

```
procedure CTX_REPORT.DESCRIBE_POLICY(
  policy_name    IN VARCHAR2,
  report         IN OUT NOCOPY CLOB,
  report_format  IN VARCHAR2 DEFAULT FMT_TEXT
);

function CTX_REPORT.DESCRIBE_POLICY(
  policy_name    IN VARCHAR2,
  report_format  IN VARCHAR2 DEFAULT FMT_TEXT
) return CLOB;
```

report
Specify the `CLOB` locator to which to write the report.

If `report` is NULL, a session-duration temporary `CLOB` will be created and returned. It is the caller's responsibility to free this temporary `CLOB` as needed.

The `report CLOB` will be truncated before `report` is generated, so any existing contents will be overwritten by this call.

report_format
Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values `CTX_REPORT.FMT_TEXT` or `CTX_REPORT.FMT_XML`.

policy_name
Specify the name of the policy to describe.

CREATE_INDEX_SCRIPT

Creates a SQL*Plus script which will create a text index that duplicates the named text index.

The created script will include creation of preferences identical to those used in the named text index. However, the names of the preferences will be different.

You can call this operation as a procedure with an `IN OUT CLOB` parameter or as a function that returns the report as a `CLOB`.

Syntax

```
procedure CTX_REPORT.CREATE_INDEX_SCRIPT(
  index_name      in varchar2,
  report          in out nocopy clob,
  prefname_prefix in varchar2 default null
);

function CTX_REPORT.CREATE_INDEX_SCRIPT(
  index_name      in varchar2,
  prefname_prefix in varchar2 default null
) return clob;
```

index_name
Specify the name of the index.

report
Specify the `CLOB` locator to which to write the script.

If `report` is NULL, a session-duration temporary `CLOB` will be created and returned. It is the caller's responsibility to free this temporary `CLOB` as needed.

The `report CLOB` will be truncated before report is generated, so any existing contents will be overwritten by this call.

prefname_prefix
Specify optional prefix to use for preference names.

If `prefname_prefix` is omitted or NULL, index name will be used. The `prefname_prefix` follows index length restrictions.

Notes

`CTX_REPORT.CREATE_INDEX_SCRIPT` will also generate necessary `FILTER BY` and `ORDER BY` clauses for `CREATE INDEX` statements.

Related Topics

"CREATE INDEX" on page 1-36.

CREATE_POLICY_SCRIPT

Creates a SQL*Plus script which will create a text policy that duplicates the named text policy.

The created script will include creation of preferences identical to those used in the named text policy.

You can call this operation as a procedure with an `IN OUT CLOB` parameter or as a function that returns the report as a `CLOB`.

Syntax

```
procedure CTX_REPORT.CREATE_POLICY_SCRIPT(
  policy_name      in varchar2,
  report           in out nocopy clob,
  prefname_prefix  in varchar2 default null
);

function CTX_REPORT.CREATE_POLICY_SCRIPT(
  policy_name      in varchar2,
  prefname_prefix  in varchar2 default null
) return clob;
```

policy_name
Specify the name of the policy.

report
Specify the locator to which to write the script.

If `report` is NULL, a session-duration temporary `CLOB` will be created and returned. It is the caller's responsibility to free this temporary `CLOB` as needed.

The `report CLOB` will be truncated before report is generated, so any existing contents will be overwritten by this call.

prefname_prefix
Specify the optional prefix to use for preference names. If `prefname_prefix` is omitted or NULL, policy name will be used. `prefname_prefix` follows policy length restrictions.

INDEX_SIZE

Creates a report showing the internal objects of the text index or text index partition, and their tablespaces, allocated, and used sizes.

You can call this operation as a procedure with an IN OUT CLOB parameter, or as a function that returns the report as a CLOB.

Syntax

```
procedure CTX_REPORT.INDEX_SIZE(
  index_name     IN VARCHAR2,
  report         IN OUT NOCOPY CLOB,
  part_name      IN VARCHAR2 DEFAULT NULL,
  report_format  IN VARCHAR2 DEFAULT FMT_TEXT
);

function CTX_REPORT.INDEX_SIZE(
  index_name     IN VARCHAR2,
  part_name      IN VARCHAR2 DEFAULT NULL,
  report_format  IN VARCHAR2 DEFAULT FMT_TEXT
) return clob;
```

index_name
Specify the name of the index to describe

report
Specify the CLOB locator to which to write the report.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call

part_name
Specify the name of the index partition (optional). If part_name is NULL, and the index is a local partitioned text index, then all objects of all partitions will be displayed. If part_name is provided, then only the objects of a particular partition will be displayed.

report_format
Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX_REPORT.FMT_TEXT or CTX_REPORT.FMT_XML.

Notes

CTX_REPORT.INDEX_SIZE will also output information on dr$*indexname*$S table.

Related Topics

"CREATE INDEX" on page 1-36

Table 2–35, " BASIC_STORAGE Attributes" on page 2-65

INDEX_STATS

Creates a report showing various calculated statistics about the text index.

This procedure fully scans the text index tables, so it may take a long time to run for large indexes.

Syntax

```
procedure ctx_report.index_stats(
  index_name     IN VARCHAR2,
  report         IN OUT NOCOPY CLOB,
  part_name      IN VARCHAR2 DEFAULT NULL,
  frag_stats     IN BOOLEAN DEFAULT TRUE,
  list_size      IN NUMBER DEFAULT 100,
  report_format  IN VARCHAR2 DEFAULT FMT_TEXT,
  stat_type      IN VARCHAR2 DEFAULT NULL
);
```

index_name
Specify the name of the index to describe. This must be a CONTEXT index.

report
Specify the CLOB locator to which to write the report.If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

part_name
Specify the name of the index partition. If the index is a local partitioned index, then part_name must be provided. INDEX_STATS will calculate the statistics for that index partition.

frag_stats
Specify TRUE to calculate fragmentation statistics. If frag_stats is FALSE, the report will not show any statistics relating to size of index data. However, the operation should take less time and resources to calculate the token statistics.

list_size
Specify the number of elements in each compiled list. list_size has a maximum value of 1000.

report_format
Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX_REPORT.FMT_TEXT or CTX_REPORT.FMT_XML.

stat_type
Specify the estimated statistics to output. If this parameter is set, then frag_stats is ignored. The possible values are:

EST_FRAG_STATS	Get the estimated fragmentation stats for the index. When this type is given list_size is ignored.

EST_FREQUENT_TOKENS	Get the estimated frequently queried tokens for the index. You can give a value of up to 100 for `list_size`.
EST_TOKENS_TO_OPTIMIZE	Show best tokens to optimize, based on frequency of querying and fragmentation. You can give a value of up to 100 for `list_size`.
EST_SLOWEST_QUERIES	Show slowest running queries for the index. You can give a value of up to 100 for `list_size`.

Note: The estimated statistics for `stat_type` is only available if `query_stats` is enabled. See CTX_OUTPUT.ENABLE_QUERY_STATS and CTX_OUTPUT.DISABLE_QUERY_STATS.

Example

```
create table output (result CLOB);

  declare
    x clob := null;
  begin
    ctx_report.index_stats('tdrbprx21',x);
    insert into output values (x);
    commit;
    dbms_lob.freetemporary(x);
  end;
  /

set long 32000
set head off
set pagesize 10000
select * from output;
```

The following sample output is for `INDEX_STATS` on a context index. This report has been truncated for clarity. It shows some of the token statistics and all of the fragmentation statistics.

The fragmentation statistics are at the end of the report. It tells you optimal row fragmentation, an estimated amount of garbage data in the index, and a list of the most fragmented tokens. Running `CTX_DDL.OPTIMIZE_INDEX` cleans up the index.

```
================================================================
              STATISTICS FOR "DR_TEST"."TDRBPRX21"
================================================================

indexed documents:                                          53
allocated docids:                                           68
$I rows:                                                16,259

----------------------------------------------------------------
                        TOKEN STATISTICS
----------------------------------------------------------------

unique tokens:                                          13,445
average $I rows for each token:                           1.21
tokens with most $I rows:
  telecommunications industry (THEME)                        6
  science and technology (THEME)                             6
  EMAIL (FIELD SECTION "SOURCE")                             6
```

```
  DEC (FIELD SECTION "TIMESTAMP")                          6
  electronic mail (THEME)                                  6
  computer networking (THEME)                              6
  communications (THEME)                                   6
  95 (FIELD SECTION "TIMESTAMP")                           6
  15 (FIELD SECTION "TIMESTAMP")                           6
  HEADLINE (ZONE SECTION)                                  6

average size for each token:                               8
tokens with largest size:
  T (NORMAL)                                             405
  SAID (NORMAL)                                          313
  HEADLINE (ZONE SECTION)                                272
  NEW (NORMAL)                                           267
  I (NORMAL)                                             230
  MILLION (PREFIX)                                       222
  D (NORMAL)                                             219
  MILLION (NORMAL)                                       215
  U (NORMAL)                                             192
  DEC (FIELD SECTION "TIMESTAMP")                        186

average frequency for each token:                       2.00
most frequent tokens:
  HEADLINE (ZONE SECTION)                                 68
  DEC (FIELD SECTION "TIMESTAMP")                         62
  95 (FIELD SECTION "TIMESTAMP")                          62
  15 (FIELD SECTION "TIMESTAMP")                          62
  T (NORMAL)                                              61
  D (NORMAL)                                              59
  881115 (THEME)                                          58
  881115 (NORMAL)                                         58
  I (NORMAL)                                              55
  geography (THEME)                                       52

token statistics by type:
  token type:                                        NORMAL
    unique tokens:                                    6,344
    total rows:                                       7,631
    average rows:                                      1.20
    total size:                          67,445 (65.86 KB)
    average size:                                        11
    average frequency:                                2.33
    most frequent tokens:
      T                                                  61
      D                                                  59
      881115                                             58
      I                                                  55
      SAID                                               45
      C                                                  43
      NEW                                                36
      MILLION                                            32
      FIRST                                              28
      COMPANY                                            27

  token type:                                        THEME
    unique tokens:                                    4,563
    total rows:                                       5,523
    average rows:                                      1.21
    total size:                          21,930 (21.42 KB)
    average size:                                         5
```

```
    average frequency:                                  2.40
    most frequent tokens:
      881115                                              58
      political geography                                 52
      geography                                           52
      United States                                       51
      business and economics                              50
      abstract ideas and concepts                         48
      North America                                       48
      science and technology                              46
      NKS                                                 34
      nulls                                               34
```

The fragmentation portion of this report is as follows:

```
------------------------------------------------------------------
                    FRAGMENTATION STATISTICS
------------------------------------------------------------------

total size of $I data:                    116,772 (114.04 KB)

$I rows:                                           16,259
estimated $I rows if optimal:                      13,445
estimated row fragmentation:                          17 %

garbage docids:                                       15
estimated garbage size:                    21,379 (20.88 KB)

most fragmented tokens:
  telecommunications industry (THEME)                 83 %
  science and technology (THEME)                      83 %
  EMAIL (FIELD SECTION "SOURCE")                      83 %
  DEC (FIELD SECTION "TIMESTAMP")                     83 %
  electronic mail (THEME)                             83 %
  computer networking (THEME)                         83 %
  communications (THEME)                              83 %
  95 (FIELD SECTION "TIMESTAMP")                      83 %
  HEADLINE (ZONE SECTION)                             83 %
  15 (FIELD SECTION "TIMESTAMP")                      83 %
```

Notes

These metadata are available only when QUERY_STATS is turned on for the index: estimated fragmentation stats, estimated frequently queried tokens, estimated most fragmented frequently queried token, and estimated slowest running queries for the specified index.

CTX_REPORT.INDEX_STATS will also output information on dr$*indexname*$S table, which is the section data, or SDATA, table.

Related Topics

CTX_OUTPUT.ENABLE_QUERY_STATS on page 9-6

CTX_OUTPUT.DISABLE_QUERY_STATS on page 9-5.

CREATE INDEX on page 1-36.

Table 2–35, " BASIC_STORAGE Attributes" on page 2-65.

QUERY_LOG_SUMMARY

Obtain a report of logged queries.

QUERY_LOG_SUMMARY enables you to analyze queries you have logged. For example, suppose you have an application that searches a database of large animals, and your analysis of queries against it shows that users are continually searching for the word *mouse*; this analysis might induce you to rewrite your application so that a search for *mouse* redirects the user to a database for small animals instead of simply returning an unsuccessful search.

With query analysis, you can find out the following:

- Which queries were made
- Which queries were successful
- Which queries were unsuccessful
- How many times each query was made

You can combine these factors in various ways, such as determining the 50 most frequent unsuccessful queries made by your application.

Query logging is begun with CTX_OUTPUT.START_QUERY_LOG and terminated with CTX_OUTPUT.END_QUERY_LOG.

Note: You must connect as CTXSYS to use CTX_REPORT.QUERY_LOG_SUMMARY.

See Also: "START_QUERY_LOG" on page 9-16 and "END_QUERY_LOG" on page 9-8

Syntax

```
procedure CTX_REPORT.QUERY_LOG_SUMMARY(
   logfile       IN VARCHAR2,
   indexname     IN VARCHAR2 DEFAULT NULL,
   result_table  IN OUT NOCOPY QUERY_TABLE,
   row_num       IN NUMBER,
   most_freq     IN BOOLEAN DEFAULT TRUE,
   has_hit       IN BOOLEAN DEFAULT TRUE
);
```

logfile

Specify the name of the logfile that contains the queries.

indexname

Specify the name of the context index for which you want the summary report. If you specify NULL, the procedure provides a summary report for all context indexes.

result_table

Specify the name of the in-memory table of type TABLE OF RECORD where the results of the QUERY_LOG_SUMMARY are to go. The default is the location specified by the system parameter LOG_DIRECTORY.

row_num
The number of rows of results from QUERY_LOG_SUMMARY to be reported into the table named by *restab*. For example, if this is number is 10, *most_freq* is TRUE, and *has_hit* is TRUE, then the procedure returns the 10 most frequent queries that were successful (that is, returned hits).

most_freq
Specify whether QUERY_LOG_SUMMARY should return the most frequent or least frequent queries. The default is most frequent queries. If *most_freq* is set to FALSE, the procedure returns the least successful queries.

has_hit
Specify whether QUERY_LOG_SUMMARY should return queries that are successful (that is, that generate hits) or unsuccessful queries. The default is to count successful queries; set *has_hit* to FALSE to return unsuccessful queries.

Example

The following example shows how a query log can be used.

First connect as CTXSYS. Then create and populate two tables, and then create an index for each:

```
create table qlogtab1 (tk number primary key, text varchar2(2000));
insert into qlogtab1 values(1, 'The Roman name for France was Gaul.');
insert into qlogtab1 values(2, 'The Tour de France is held each summer.');
insert into qlogtab1 values(3, 'Jacques Anatole Thibault took the pen name Anatole France.');
create index idx_qlog1 on qlogtab1(text) indextype is ctxsys.context;
create table qlogtab2 (tk number primary key, text varchar2(2000));
insert into qlogtab2 values(1, 'The Great Wall of China is about 2400 kilometers long');
insert into qlogtab2 values(2, 'Soccer dates back at least to 217 C.E.');
insert into qlogtab2 values(3, 'The Corn Palace is a tourist attraction in South Dakota.');
create index idx_qlog2 on qlogtab2(text) indextype is ctxsys.context;
```

Turn on query logging, creating a log called query_log:

```
exec ctx_output.start_query_log('query.log');
```

Now make some queries (some of which will be unsuccessful):

```
select text from qlogtab1 where contains(text, 'France',1)>0;
select text from qlogtab1 where contains(text, 'cheese',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizard',1)>0;
select text from qlogtab2 where contains(text, 'Corn Palace',1)>0;
select text from qlogtab2 where contains(text, 'China',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizards',1)>0;
select text from qlogtab2 where contains(text, 'South Dakota',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizard',1)>0;
select text from qlogtab2 where contains(text, 'China',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizard',1)>0;
select text from qlogtab2 where contains(text, 'company',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizard',1)>0;
select text from qlogtab1 where contains(text, 'France',1)>0;
select text from qlogtab1 where contains(text, 'database',1)>0;
select text from qlogtab2 where contains(text, 'high-tech',1)>0;
select text from qlogtab1 where contains(text, 'database',1)>0;
select text from qlogtab1 where contains(text, 'France',1)>0;
select text from qlogtab1 where contains(text, 'Japan',1)>0;
select text from qlogtab1 where contains(text, 'Egypt',1)>0;
select text from qlogtab1 where contains(text, 'Argentina',1)>0;
```

```
select text from qlogtab1 where contains(text, 'Argentina',1)>0;
select text from qlogtab1 where contains(text, 'Argentina',1)>0;
select text from qlogtab1 where contains(text, 'Japan',1)>0;
select text from qlogtab1 where contains(text, 'Egypt',1)>0;
select text from qlogtab1 where contains(text, 'Air Shuttle',1)>0;
select text from qlogtab1 where contains(text, 'Argentina',1)>0;
```

With the querying over, turn query logging off:

```
exec ctx_output.end_query_log;
```

Use QUERY_LOG_SUMMARY to get query reports. In the first instance, you ask to see the three most frequent queries that return successfully. First declare the results table (the_queries).

```
set serveroutput on;
declare
   the_queries ctx_report.query_table;
begin
   ctx_report.query_log_summary('query.log', null, the_queries,
                  row_num=>3, most_freq=>TRUE, has_hit=>TRUE);
   dbms_output.put_line('The 3 most frequent queries returning hits');
   dbms_output.put_line('number of times  query string');
   for i in 1..the_queries.count loop
     dbms_output.put_line(the_queries(i).times||'               '||the_queries(i).query);
   end loop;
end;
/
```

This returns the following:

```
TThe 3 most frequent queries returning hits
number of times  query string
3                France
2                China
1                Corn Palace
```

Next, look for the three most frequent queries on idx_qlog1 that were successful.

```
declare
   the_queries ctx_report.query_table;
begin
   ctx_report.query_log_summary('query.log', 'idx_qlog1', the_queries,
                  row_num=>3, most_freq=>TRUE, has_hit=>TRUE);
   dbms_output.put_line('The 3 most frequent queries returning hits for index idx_qlog1');
   dbms_output.put_line('number of times  query string');
   for i in 1..the_queries.count loop
     dbms_output.put_line(the_queries(i).times||'               '||the_queries(i).query);
   end loop;
end;
/
```

Because only the queries for *France* were successful, ctx_report.query_log_summary returns the following:

```
The 3 most frequent queries returning hits for index idx_qlog1
number of times  query string
3                France
```

Lastly, ask to see the three least frequent queries that returned no hits (that is, queries that were unsuccessful and called infrequently). In this case, you are interested in queries on both context indexes, so you set the indexname parameter to NULL.

```
declare
   the_queries ctx_report.query_table;
begin
   ctx_report.query_log_summary('query.log', null, the_queries, row_num=>3,
                  most_freq=>FALSE, has_hit=>FALSE);
   dbms_output.put_line('The 3 least frequent queries returning no hit');
   dbms_output.put_line('number of times  query string');
   for i in 1..the_queries.count loop
     dbms_output.put_line(the_queries(i).times||'                '||the_queries(i).query);
   end loop;
end;
/
```

This returns the following results:

```
The 3 least frequent queries returning no hit
number of times  query string
1                high-tech
1                company
1                cheese
```

Argentina and *Japan* do not make this list, because they are queried more than once, while *Corn Palace* does not make this list because it is successfully queried.

TOKEN_INFO

Creates a report showing the information for a token, decoded. This procedure will fully scan the info for a token, so it may take a long time to run for really large tokens.

You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

Syntax

```
procedure CTX_REPORT.TOKEN_INFO(
  index_name      IN VARCHAR2,
  report          IN OUT NOCOPY CLOB,
  token           IN VARCHAR2,
  token_type      IN NUMBER,
  part_name       IN VARCHAR2 DEFAULT NULL,
  raw_info        IN BOOLEAN  DEFAULT FALSE,
  decoded_info    IN BOOLEAN  DEFAULT TRUE,
  report_format   IN VARCHAR2 DEFAULT FMT_TEXT
);

function CTX_REPORT.TOKEN_INFO(
  index_name      IN VARCHAR2,
  token           IN VARCHAR2,
  token_type      IN NUMBER,
  part_name       IN VARCHAR2 DEFAULT NULL,
  raw_info        IN VARCHAR2 DEFAULT 'N',
  decoded_info    IN VARCHAR2 DEFAULT 'Y',
  report_format   IN VARCHAR2 DEFAULT FMT_TEXT
) return clob;
```

index_name
Specify the name of the index.

report
Specify the CLOB locator to which to write the report.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call token may be case-sensitive, depending on the passed-in token type.

token
Specify the token text.

token_type
Specify the token type. You can use a number returned by the TOKEN_TYPE function. THEME, ZONE, ATTR, PATH, and PATH ATTR tokens are case-sensitive.

Everything else gets passed through the lexer, so if the index's lexer is case-sensitive, the token input is case-sensitive.

part_name
Specify the name of the index partition.

If the index is a local partitioned index, then part_name must be provided. TOKEN_INFO will apply to just that index partition.

raw_info
Specify TRUE to include a hex dump of the index data. If raw_info is TRUE, the report will include a hex dump of the raw data in the token_info column.

decoded_info
Specify decode and include docid and offset data. If decoded_info is FALSE, CTX_REPORT will not attempt to decode the token information. This is useful when you just want a dump of data.

report_format
Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX_REPORT.FMT_TEXT or CTX_REPORT.FMT_XML.

TOKEN_TYPE

This is a helper function which translates an English name into a numeric token type. This is suitable for use with `token_info`, or any other `CTX` API which takes in a `token_type`.

```
function token_type(
  index_name in varchar2,
  type_name  in varchar2
) return number;

TOKEN_TYPE_TEXT      constant number := 0;
TOKEN_TYPE_THEME     constant number := 1;
TOKEN_TYPE_ZONE_SEC  constant number := 2;
TOKEN_TYPE_ORIG      constant number := 3,
TOKEN_TYPE_ATTR_TEXT constant number := 4;
TOKEN_TYPE_ATTR_SEC  constant number := 5;
TOKEN_TYPE_PREFIX    constant number := 6;
TOKEN_TYPE_PATH_SEC  constant number := 7;
TOKEN_TYPE_PATH_ATTR constant number := 8;
TOKEN_TYPE_STEM      constant number := 9;
```

index_name
Specify the name of the index.

type_name
Specify an English name for `token_type`. The following strings are legal input. All input is case-insensitive.

Input	Meaning	Type Returned
`TEXT`	Normal text token.	0
`THEME`	Theme token.	1
`ZONE SEC`	Zone token.	2
`ORIGINAL`	Original form token	3
`ATTR TEXT`	Text that occurs in attribute.	4
`ATTR SEC`	Attribute section.	5
`PREFIX`	Prefix token.	6
`PATH SEC`	Path section.	7
`PATH ATTR`	Path attribute section.	8
`STEM`	Stem form token.	9
`FIELD <name> TEXT`	Text token in field section <name>	16-79
`FIELD <name> PREFIX`	Prefix token in field section <name>	616-916
`FIELD <name> STEM`	Stem token in field section <name>	916-979
`NDATA <name>`	`NDATA`-type token	200-299
`TOKEN_TYPE_ATTR_TXT_PFIX`	Attribute text prefix.	604
`TOKEN_TYPE_ATTR_TXT_STEM`	Attribute text stem.	904

For FIELD types, the index metadata needs to be read, so if you are going to be calling this a lot for such things, you might want to consider caching the values in local variables rather than calling token_type over and over again.

The constant types (0 - 9) also have constants in this package defined.

Notes

To get token types for MDATA tokens, do not use CTX_REPORT.TOKEN_TYPE; use the MDATA operator instead. (See "MDATA" on page 3-28.) The syntax to use is 'MDATA *fieldname*'.

Example

```
typenum := ctx_report.token_type('myindex', 'field author text');
```

12

CTX_THES Package

This chapter provides reference information for using the CTX_THES package to manage and browse thesauri. These thesaurus functions are based on the ISO-2788 and ANSI Z39.19 standards except where noted.

Knowing how information is stored in your thesaurus helps in writing queries with thesaurus operators. You can also use a thesaurus to extend the knowledge base, which is used for ABOUT queries in English and French and for generating document themes.

CTX_THES contains the following stored procedures and functions:

Name	Description
ALTER_PHRASE	Alters thesaurus phrase.
ALTER_THESAURUS	Renames or truncates a thesaurus.
BT	Returns all broader terms of a phrase.
BTG	Returns all broader terms generic of a phrase.
BTI	Returns all broader terms instance of a phrase.
BTP	Returns all broader terms partitive of a phrase.
CREATE_PHRASE	Adds a phrase to the specified thesaurus.
CREATE_RELATION	Creates a relation between two phrases.
CREATE_THESAURUS	Creates the specified thesaurus.
CREATE_TRANSLATION	Creates a new translation for a phrase.
DROP_PHRASE	Removes a phrase from thesaurus.
DROP_RELATION	Removes a relation between two phrases.
DROP_THESAURUS	Drops the specified thesaurus from the thesaurus tables.
DROP_TRANSLATION	Drops a translation for a phrase.
HAS_RELATION	Tests for the existence of a thesaurus relation.
NT	Returns all narrower terms of a phrase.
NTG	Returns all narrower terms generic of a phrase.
NTI	Returns all narrower terms instance of a phrase.
NTP	Returns all narrower terms partitive of a phrase.
OUTPUT_STYLE	Sets the output style for the expansion functions.
PT	Returns the preferred term of a phrase.

Name	Description
RT	Returns the related terms of a phrase
SN	Returns scope note for phrase.
SYN	Returns the synonym terms of a phrase
THES_TT	Returns all top terms for phrase.
TR	Returns the foreign equivalent of a phrase.
TRSYN	Returns the foreign equivalent of a phrase, synonyms of the phrase, and foreign equivalent of the synonyms.
TT	Returns the top term of a phrase.
UPDATE_TRANSLATION	Updates an existing translation.

See Also: Chapter 3, "Oracle Text CONTAINS Query Operators" for more information about the thesaurus operators.

ALTER_PHRASE

Alters an existing phrase in the thesaurus. Only CTXSYS or thesaurus owner can alter a phrase.

Syntax

```
CTX_THES.ALTER_PHRASE(tname      in varchar2,
                      phrase     in varchar2,
                      op         in varchar2,
                      operand    in varchar2 default null);
```

tname
Specify the thesaurus name.

phrase
Specify a phrase to alter.

op
Specify the alter operation as a string or symbol. You can specify one of the following operations with the op and operand pair:'

op	meaning	operand
RENAME or CTX_THES.OP_RENAME	Rename phrase. If the new phrase already exists in the thesaurus, this procedure raises an exception.	Specify a new phrase. You can include qualifiers to change, add, or remove qualifiers from phrases.
PT or CTX_THES.OP_PT	Make phrase the preferred term. Existing preferred terms in the synonym ring becomes non-preferred synonym.	(none)
SN or CTX_THES.OP_SN	Change the scope note on the phrase.	Specify a new scope note.

operand
Specify an argument to the alter operation. See table for op.

Examples

Correct misspelled word in thesaurus:

```
ctx_thes.alter_phrase('thes1', 'tee', 'rename', 'tea');
```

Remove qualifier from mercury (metal):

```
ctx_thes.alter_phrase('thes1', 'mercury (metal)', 'rename', 'mercury');
```

Add qualifier to mercury:

```
ctx_thes.alter_phrase('thes1', 'mercury', 'rename', 'mercury (planet)');
```

Make Kowalski the preferred term in its synonym ring:

```
ctx_thes.alter_phrase('thes1', 'Kowalski', 'pt');
```

Change scope note for view cameras:

```
ctx_thes.alter_phrase('thes1', 'view cameras', 'sn', 'Cameras with lens
focusing');
```

ALTER_THESAURUS

Use this procedure to rename or truncate an existing thesaurus. Only the thesaurus owner or CTXSYS can invoke this function on a given thesaurus.

Syntax

```
CTX_THES.ALTER_THESAURUS(tname     in   varchar2,
                         op        in   varchar2,
                         operand   in   varchar2 default null);
```

tname
Specify the thesaurus name.

op
Specify the alter operation as a string or symbol. You can specify one of two operations:

op	Meaning	operand
RENAME or CTX_THES.OP_RENAME	Rename thesaurus. Returns an error if the new name already exists.	Specify a new thesaurus name.
TRUNCATE or CTX_THES.OP_TRUNCATE	Truncate thesaurus.	None.

operand
Specify the argument to the alter operation. See table for op.

Examples

Rename thesaurus THES1 to MEDICAL:

```
ctx_thes.alter_thesaurus('thes1', 'rename', 'medical');
```

or

```
ctx_thes.alter_thesaurus('thes1', ctx_thes.op_rename, 'medical');
```

You can use symbols for any op argument, but all further examples will use strings.

Remove all phrases and relations from thesaurus THES1:

```
ctx_thes.alter_thesaurus('thes1', 'truncate');
```

BT

This function returns all broader terms of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.BT(restab IN OUT NOCOPY EXP_TAB,
            phrase IN VARCHAR2,
            lvl    IN NUMBER DEFAULT 1,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.BT(phrase IN VARCHAR2,
            lvl    IN NUMBER DEFAULT 1,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type `EXP_TAB` which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about `EXP_TAB`

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.

tname

Specify a thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of broader terms in the form:

```
{bt1}|{bt2}|{bt3} ...
```

Example

String Result

Consider a thesaurus named `MY_THES` that has an entry for *cat* as follows:

```
cat
   BT1 feline
```

```
    BT2 mammal
      BT3 vertebrate
        BT4 animal
```

To look up the broader terms for *cat* up to two levels, enter the following statements:

```
set serveroutput on

declare
  terms varchar2(2000);
begin
  terms := ctx_thes.bt('CAT', 2, 'MY_THES');
  dbms_output.put_line('The broader expansion for CAT is: '||terms);
end;
```

This code produces the following output:

```
The broader expansion for CAT is: {cat}|{feline}|{mammal}
```

Table Result

The following code does an broader term lookup for *white wolf* using the table result:

```
set serveroutput on

declare
  xtab ctx_thes.exp_tab;
begin
  ctx_thes.bt(xtab, 'white wolf', 2, 'my_thesaurus');
  for i in 1..xtab.count loop
    dbms_output.put_line(xtab(i).rel||' '||xtab(i).phrase);
  end loop;
end;
```

This code produces the following output:

```
PHRASE WHITE WOLF
BT WOLF
BT CANINE
BT ANIMAL
```

Related Topics

OUTPUT_STYLE

Broader Term (BT, BTG, BTP, BTI) Operators in Chapter 3, "Oracle Text CONTAINS Query Operators"

BTG

This function returns all broader terms generic of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.BTG(restab IN OUT NOCOPY EXP_TAB,
             phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.BTG(phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about EXP_TAB.

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.

tname

Specify thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of broader terms generic in the form:

```
{bt1}|{bt2}|{bt3} ...
```

Example

To look up the broader terms generic for *cat* up to two levels, enter the following statements:

```
set serveroutput on
declare
  terms varchar2(2000);
begin
```

```
  terms := ctx_thes.btg('CAT', 2, 'MY_THES');
  dbms_output.put_line('the broader expansion for CAT is: '||terms);
end;
```

Related Topics

OUTPUT_STYLE

Broader Term (BT, BTG, BTP, BTI) Operators in Chapter 3, "Oracle Text CONTAINS Query Operators"

BTI

This function returns all broader terms instance of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.BTI(restab IN OUT NOCOPY EXP_TAB,
             phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.BTI(phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about EXP_TAB.

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.

tname

Specify a thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of broader terms instance in the form:

```
{bt1}|{bt2}|{bt3} ...
```

Example

To look up the broader terms instance for *cat* up to two levels, enter the following statements:

```
set serveroutput on
declare
  terms varchar2(2000);
begin
```

```
  terms := ctx_thes.bti('CAT', 2, 'MY_THES');
  dbms_output.put_line('the broader expansion for CAT is: '||terms);
end;
```

Related Topics

OUTPUT_STYLE

Broader Term (BT, BTG, BTP, BTI) Operators in Chapter 3, "Oracle Text CONTAINS Query Operators"

BTP

This function returns all broader terms partitive of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.BTP(restab IN OUT NOCOPY EXP_TAB,
             phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.BTP(phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type `EXP_TAB` which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about `EXP_TAB`.

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.

tname

Specify a thesaurus name. If not specified, the system default thesaurus is used.

Returns

This function returns a string of broader terms in the form:

```
{bt1}|{bt2}|{bt3} ...
```

Example

To look up the two broader terms partitive for *cat*, enter the following statements:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.btp('CAT', 2, 'MY_THES');
  dbms_output.put_line('the broader expansion for CAT is: '||terms);
```

```
end;
```

Related Topics

OUTPUT_STYLE

Broader Term (BT, BTG, BTP, BTI) Operators in Chapter 3, "Oracle Text CONTAINS Query Operators"

CREATE_PHRASE

The CREATE_PHRASE procedure adds a new phrase to the specified thesaurus.

> **Note:** Even though you can create thesaurus relations with this procedure, Oracle recommends that you use CTX_THES.CREATE_RELATION rather than CTX_THES.CREATE_PHRASE to create relations in a thesaurus.

Syntax

```
CTX_THES.CREATE_PHRASE(tname   IN VARCHAR2,
                       phrase  IN VARCHAR2,
                       rel     IN VARCHAR2 DEFAULT NULL,
                       relname IN VARCHAR2 DEFAULT NULL);
```

tname
Specify the name of the thesaurus in which the new phrase is added or the existing phrase is located.

phrase
Specify the phrase to be added to a thesaurus or the phrase for which a new relationship is created.

rel
Specify the new relationship between *phrase* and *relname*. This parameter is supported only for backward compatibility. Use CTX_THES.CREATE_RELATION to create new relations in a thesaurus.

relname
Specify the existing phrase that is related to *phrase*. This parameter is supported only for backward compatibility. Use CTX_THES.CREATE_RELATION to create new relations in a thesaurus.

Returns

The ID for the entry.

Example

In this example, two new phrases (*os* and *operating system*) are created in a thesaurus named tech_thes.

```
begin
   ctx_thes.create_phrase('tech_thes','os');
   ctx_thes.create_phrase('tech_thes','operating system');
end;
```

CREATE_RELATION

Creates a relation between two phrases in the thesaurus. The synonym ring is limited in length to about 4000 synonyms, depending on word length.

> **Note:** Oracle recommends that you use `CTX_THES.CREATE_RELATION` rather than `CTX_THES.CREATE_PHRASE` to create relations in a thesaurus.

Only thesaurus owner and `CTXSYS` can invoke this procedure on a given thesaurus.

Syntax

```
CTX_THES.CREATE_RELATION(tname     in    varchar2,
                         phrase    in    varchar2,
                         rel       in    varchar2,
                         relphrase in    varchar2);
```

tname
Specify the thesaurus name

phrase
Specify the phrase to alter or create. If `phrase` is a disambiguated homograph, you must specify the qualifier. If `phrase` does not exist in the thesaurus, it is created.

rel
Specify the relation to create.The relation is from `phrase` to `relphrase`. You can specify one of the following relations:

relation	meaning	relphrase
BT*/NT*	Add hierarchical relation.	Specify the related phrase. The relationship is interpreted from phrase to relphrase.
RT	Add associative relation.	Specify the phrase to associate.
SYN	Add phrase to a synonym ring.	Specify an existing phrase in the synonym ring.
Specify language	Add translation for a phrase.	Specify a new translation phrase.

relphrase
Specify the related phrase. If relphrase does not exist in tname, relphrase is created. See table for rel.

Notes

The relation you specify for rel is interpreted as from phrase to relphrase. For example, consider dog with broader term animal:

```
dog
  BT animal
```

To add this relation, specify the arguments as follows:

```
begin
```

```
CTX_THES.CREATE_RELATION('thes','dog','BT','animal');
end;
```

> **Note:** The order in which you specify arguments for CTX_THES.CREATE_RELATION is different from the order you specify them with CTX_THES.CREATE_PHRASE.

Examples

Create relation VEHICLE NT CAR:

```
ctx_thes.create_relation('thes1', 'vehicle', 'NT', 'car');
```

Create Japanese translation for *you*:

```
ctx_thes.create_relation('thes1', 'you', 'JAPANESE:', 'kimi');
```

CREATE_THESAURUS

The CREATE_THESAURUS procedure creates an empty thesaurus with the specified name in the thesaurus tables.

Syntax

```
CTX_THES.CREATE_THESAURUS(name           IN VARCHAR2,
                          casesens       IN BOOLEAN DEFAULT FALSE);
```

name
Specify the name of the thesaurus to be created. The name of the thesaurus must be unique. If a thesaurus with the specified name already exists, CREATE_THESAURUS returns an error and does not create the thesaurus.

casesens
Specify whether the thesaurus to be created is case-sensitive. If casesens is *true*, Oracle Text retains the cases of all terms entered in the specified thesaurus. As a result, queries that use the thesaurus are case-sensitive.

Example

```
begin
   ctx_thes.create_thesaurus('tech_thes', FALSE);
end;
```

CREATE_TRANSLATION

Use this procedure to create a new translation for a phrase in a specified language.

Syntax

```
CTX_THES.CREATE_TRANSLATION(tname       in    varchar2,
                            phrase      in    varchar2,
                            language    in    varchar2,
                            translation in    varchar2);
```

tname
Specify the name of the thesaurus, using no more than 30 characters.

phrase
Specify the phrase in the thesaurus to which to add a translation. Phrase must already exist in the thesaurus, or an error is raised.

language
Specify the language of the translation, using no more than 10 characters.

translation
Specify the translated term, using no more than 256 characters.

If a translation for this phrase already exists, this new translation is added without removing that original translation, so long as that original translation is not the same. Adding the same translation twice results in an error.

Example

The following code adds the Spanish translation for *dog* to *my_thes*:

```
begin
   ctx_thes.create_translation('my_thes', 'dog', 'SPANISH', 'PERRO');
end;
```

DROP_PHRASE

Removes a phrase from the thesaurus. Only thesaurus owner and CTXSYS can invoke this procedure on a given thesaurus.

Syntax

```
CTX_THES.DROP_PHRASE(tname      in varchar2,
                     phrase     in varchar2);
```

tname
Specify thesaurus name.

phrase
Specify a phrase to drop. If the phrase is a disambiguated homograph, then you must include the qualifier. If the phrase does not exist in tname, then this procedure raises an exception.

BT* / NT* relations are patched around the dropped phrase. For example, if A has a BT B, and B has BT C, after B is dropped, A has BT C.

When a word has multiple broader terms, then a relationship is established for each narrower term to each broader term.

Note that BT, BTG, BTP, and BTI are separate hierarchies, so if A has BTG B, and B has BTI C, when B is dropped, there is no relation implicitly created between A and C.

RT relations are not patched. For example, if A has RT B, and B has RT C, then if B is dropped, there is no associative relation created between A and C.

Example

Assume you have the following relations defined in *mythes*:

```
wolf
   BT canine
canine
   BT animal
```

You drop phrase *canine*:

```
begin
ctx_thes.drop_phrase('mythes', 'canine');
end;
```

The resulting thesaurus is patched and looks like:

```
wolf
   BT animal
```

DROP_RELATION

Removes a relation between two phrases from the thesaurus.

> **Note:** `CTX_THES.DROP_RELATION` removes only the relation between two phrases. Phrases are never removed by this call.

Only thesaurus owner and `CTXSYS` can invoke this procedure on a given thesaurus.

Syntax

```
CTX_THES.DROP_RELATION(tname     in    varchar2,
                       phrase    in    varchar2,
                       rel       in    varchar2,
                       relphrase in    varchar2 default null);
```

tname
Specify the thesaurus name.

phrase
Specify the filing phrase.

rel
Specify the relation to drop. The relation is from phrase to relphrase. You can specify one of the following relations:

relation	meaning	relphrase
BT*/NT*	Remove hierarchical relation.	Optional specify relphrase. If not provided, all relations of that type for the phrase are removed.
RT	Remove associative relation.	Optionally specify relphrase. If not provided, all RT relations for the phrase are removed.
SYN	Remove phrase from its synonym ring.	(none)
PT	Remove preferred term designation from the phrase. The phrase remains in the synonym ring.	(none)
language	Remove a translation from a phrase.	Optionally specify relphrase. You can specify relphrase when there are multiple translations for a phrase for the language, and you want to remove just one translation. If relphrase is NULL, all translations for the phrase for the language are removed.

relphrase
Specify the related phrase.

Notes

The relation you specify for rel is interpreted as from phrase to relphrase. For example, consider dog with broader term animal:

```
dog
  BT animal
```

To remove this relation, specify the arguments as follows:

```
begin
CTX_THES.DROP_RELATION('thes','dog','BT','animal');
end;
```

You can also remove this relation using NT as follows:

```
begin
CTX_THES.DROP_RELATION('thes','animal','NT','dog');
end;
```

Example

Remove relation VEHICLE NT CAR:

```
ctx_thes.drop_relation('thes1', 'vehicle', 'NT', 'car');
```

Remove all narrower term relations for vehicle:

```
ctx_thes.drop_relation('thes1', 'vehicle', 'NT');
```

Remove Japanese translations for *me*:

```
ctx_thes.drop_relation('thes1', 'me', 'JAPANESE:');
```

Remove a specific Japanese translation for *me*:

```
ctx_thes.drop_relation('thes1', 'me', 'JAPANESE:', 'boku')
```

DROP_THESAURUS

The DROP_THESAURUS procedure deletes the specified thesaurus and all of its entries from the thesaurus tables.

Syntax

```
CTX_THES.DROP_THESAURUS(name IN VARCHAR2);
```

name
Specify the name of the thesaurus to be dropped.

Example

```
begin
ctx_thes.drop_thesaurus('tech_thes');
end;
```

DROP_TRANSLATION

Use this procedure to remove one or more translations for a phrase.

Syntax

```
CTX_THES.DROP_TRANSLATION (tname       in    varchar2,
                           phrase      in    varchar2,
                           language    in    varchar2 default null,
                           translation in    varchar2 default null);
```

tname
Specify the name of the thesaurus, using no more than 30 characters.

phrase
Specify the phrase in the thesaurus to which to remove a translation. The phrase must already exist in the thesaurus or an error is raised.

language
Optionally, specify the language of the translation, using no more than 10 characters. If not specified, the translation must also not be specified and all translations in all languages for the phrase are removed. An error is raised if the phrase has no translations.

translation
Optionally, specify the translated term to remove, using no more than 256 characters. If no such translation exists, an error is raised.

Example

The following code removes the Spanish translation for *dog*:

```
begin
   ctx_thes.drop_translation('my_thes', 'dog', 'SPANISH', 'PERRO');
end;
```

To remove all translations for dog in all languages:

```
begin
   ctx_thes.drop_translation('my_thes', 'dog');
end;
```

HAS_RELATION

Use this procedure to test that a thesaurus relation exists without actually doing the expansion. The function returns TRUE if the phrase has any of the relations in the specified list.

Syntax

```
CTX_THES.HAS_RELATION(phrase in varchar2,
                      rel in varchar2,
                      tname in varchar2 default 'DEFAULT')
  returns boolean;
```

phrase
Specify the phrase.

rel
Specify a single thesaural relation or a comma-delimited list of relations, except PT. Specify 'ANY' for any relation.

tname
Specify the thesaurus name.

Example

The following example returns TRUE if the phrase *cat* in the DEFAULT thesaurus has any broader terms or broader generic terms:

```
set serveroutput on
result boolean;

begin
 result := ctx_thes.has_relation('cat','BT,BTG');
 if (result) then dbms_output.put_line('TRUE');
  else  dbms_output.put_line('FALSE');
 end if;
end;
```

NT

This function returns all narrower terms of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.NT(restab IN OUT NOCOPY EXP_TAB,
            phrase IN VARCHAR2,
            lvl    IN NUMBER DEFAULT 1,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.NT(phrase IN VARCHAR2,
            lvl    IN NUMBER DEFAULT 1,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about EXP_TAB.

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

tname

Specify thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of narrower terms in the form:

```
{nt1}|{nt2}|{nt3} ...
```

Example

String Result

Consider a thesaurus named MY_THES that has an entry for *cat* as follows:

```
cat
 NT domestic cat
```

```
 NT wild cat
 BT mammal
mammal
 BT animal
domestic cat
 NT Persian cat
 NT Siamese cat
```

To look up the narrower terms for *cat* down to two levels, enter the following statements:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.nt('CAT', 2, 'MY_THES');
  dbms_output.put_line('the narrower expansion for CAT is: '||terms);
end;
```

This code produces the following output:

```
the narrower expansion for CAT is: {cat}|{domestic cat}|{Persian cat}|{Siamese
cat}| {wild cat}
```

Table Result

The following code does an narrower term lookup for *canine* using the table result:

```
declare
  xtab ctx_thes.exp_tab;
begin
  ctx_thes.nt(xtab, 'canine', 2, 'my_thesaurus');
  for i in 1..xtab.count loop
    dbms_output.put_line(lpad(' ', 2*xtab(i).xlevel) ||
    xtab(i).xrel || ' ' || xtab(i).xphrase);
  end loop;
end;
```

This code produces the following output:

```
PHRASE CANINE
 NT WOLF (Canis lupus)
   NT WHITE WOLF
   NT GREY WOLF
 NT DOG (Canis familiaris)
   NT PIT BULL
   NT DASCHUND
   NT CHIHUAHUA
NT HYENA (Canis mesomelas)
NT COYOTE (Canis latrans)
```

Related Topics

OUTPUT_STYLE

Narrower Term (NT, NTG, NTP, NTI) Operators in Chapter 3, "Oracle Text CONTAINS Query Operators"

NTG

This function returns all narrower terms generic of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.NTG(restab IN OUT NOCOPY EXP_TAB,
             phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.NTG(phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type `EXP_TAB` which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about `EXP_TAB`.

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of narrower terms generic in the form:

```
{nt1}|{nt2}|{nt3} ...
```

Example

To look up the narrower terms generic for *cat* down to two levels, enter the following statements:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.ntg('CAT', 2, 'MY_THES');
```

```
  dbms_output.put_line('the narrower expansion for CAT is: '||terms);
end;
```

Related Topics

OUTPUT_STYLE

Narrower Term (NT, NTG, NTP, NTI) Operators in Chapter 3, "Oracle Text CONTAINS Query Operators"

NTI

This function returns all narrower terms instance of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.NTI(restab IN OUT NOCOPY EXP_TAB,
             phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.NTI(phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type `EXP_TAB` which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about `EXP_TAB`.

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of narrower terms instance in the form:

```
{nt1}|{nt2}|{nt3} ...
```

Example

To look up the narrower terms instance for *cat* down to two levels, enter the following statements:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.nti('CAT', 2, 'MY_THES');
```

```
  dbms_output.put_line('the narrower expansion for CAT is: '||terms);
end;
```

Related Topics

OUTPUT_STYLE

Narrower Term (NT, NTG, NTP, NTI) Operators in Chapter 3, "Oracle Text CONTAINS Query Operators"

NTP

This function returns all narrower terms partitive of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.NTP(restab IN OUT NOCOPY EXP_TAB,
             phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.NTP(phrase IN VARCHAR2,
             lvl    IN NUMBER DEFAULT 1,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type `EXP_TAB` which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about `EXP_TAB`.

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of narrower terms partitive in the form:

```
{nt1}|{nt2}|{nt3} ...
```

Example

To look up the narrower terms partitive for *cat* down to two levels, enter the following statements:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.ntp('CAT', 2, 'MY_THES');
```

```
  dbms_output.put_line('the narrower expansion for CAT is: '||terms);
end;
```

Related Topics

OUTPUT_STYLE

Narrower Term (NT, NTG, NTP, NTI) Operators in Chapter 3, "Oracle Text CONTAINS Query Operators"

OUTPUT_STYLE

Sets the output style for the return string of the CTX_THES expansion functions. This procedure has no effect on the table results to the CTX_THES expansion functions.

Syntax

```
CTX_THES.OUTPUT_STYLE (
      showlevel     IN BOOLEAN DEFAULT FALSE,
      showqualify   IN BOOLEAN DEFAULT FALSE,
      showpt        IN BOOLEAN DEFAULT FALSE,
      showid        IN BOOLEAN DEFAULT FALSE
);
```

showlevel
Specify TRUE to show level in BT/NT expansions.

showqualify
Specify TRUE to show phrase qualifiers.

showpt
Specify TRUE to show preferred terms with an asterisk *.

showid
Specify TRUE to show phrase ids.

Notes

The general syntax of the return string for CTX_THES expansion functions is:

```
{pt indicator:phrase (qualifier):level:phraseid}
```

Preferred term indicator is an asterisk then a colon at the start of the phrase. The qualifier is in parentheses after a space at the end of the phrase. Level is a number.

The following is an example return string for turkey the bird:

```
*:TURKEY (BIRD):1:1234
```

PT

This function returns the preferred term of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.PT(restab IN OUT NOCOPY EXP_TAB,
            phrase IN VARCHAR2,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN varchar2;
```

Syntax 2: String Result

```
CTX_THES.PT(phrase IN VARCHAR2,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN varchar2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about EXP_TAB.

phrase

Specify a phrase to lookup in thesaurus.

tname

Specify thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns the preferred term as a string in the form:

```
{pt}
```

Example

Consider a thesaurus MY_THES with the following preferred term definition for automobile:

```
AUTOMOBILE
  PT CAR
```

To look up the preferred term for *automobile*, execute the following code:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.pt('AUTOMOBILE','MY_THES');
```

```
  dbms_output.put_line('The prefered term for automobile is: '||terms);
end;
```

Related Topics

OUTPUT_STYLE

Preferred Term (PT) Operator in Chapter 3, "Oracle Text CONTAINS Query Operators"

RT

This function returns the related terms of a term in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.RT(restab IN OUT NOCOPY EXP_TAB,
            phrase IN VARCHAR2,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.RT(phrase IN VARCHAR2,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN varchar2;
```

restab
Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about EXP_TAB.

phrase
Specify a phrase to lookup in thesaurus.

tname
Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of related terms in the form:

```
{rt1}|{rt2}|{rt3}| ...
```

Example

Consider a thesaurus MY_THES with the following related term definition for dog:

```
DOG
  RT WOLF
  RT HYENA
```

To look up the related terms for *dog*, execute the following code:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.rt('DOG','MY_THES');
  dbms_output.put_line('The related terms for dog are: '||terms);
end;
```

This codes produces the following output:

```
The related terms for dog are: {dog}|{wolf}|{hyena}
```

Related Topics

OUTPUT_STYLE

Related Term (RT) Operator in Chapter 3, "Oracle Text CONTAINS Query Operators"

SN

This function returns the scope note of the given phrase.

Syntax

```
CTX_THES.SN(phrase IN VARCHAR2,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

phrase
Specify a phrase to lookup in thesaurus.

tname
Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns the scope note as a string.

Example

```
declare
  note varchar2(80);
begin
  note := ctx_thes.sn('camera','mythes');
  dbms_output.put_line('CAMERA');
  dbms_output.put_line(' SN ' || note);
end;

sample output:

CAMERA
 SN Optical cameras
```

SYN

This function returns all synonyms of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.SYN(restab IN OUT NOCOPY EXP_TAB,
             phrase IN VARCHAR2,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.SYN(phrase IN VARCHAR2,
             tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about EXP_TAB.

phrase

Specify a phrase to lookup in thesaurus.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of the form:

```
{syn1}|{syn2}|{syn3} ...
```

Example

String Result

Consider a thesaurus named ANIMALS that has an entry for *cat* as follows:

```
CAT
  SYN KITTY
  SYN FELINE
```

To look-up the synonym for *cat* and obtain the result as a string, enter the following statements:

```
declare
  synonyms varchar2(2000);
begin
```

```
  synonyms := ctx_thes.syn('CAT','ANIMALS');
  dbms_output.put_line('the synonym expansion for CAT is: '||synonyms);
end;
```

This code produces the following output:

```
the synonym expansion for CAT is: {CAT}|{KITTY}|{FELINE}
```

Table Result

The following code looks up the synonyms for *canine* and obtains the results in a table. The contents of the table are printed to the standard output.

```
declare
  xtab ctx_thes.exp_tab;
begin
  ctx_thes.syn(xtab, 'canine', 'my_thesaurus');
  for i in 1..xtab.count loop
    dbms_output.put_line(lpad(' ', 2*xtab(i).xlevel) ||
    xtab(i).xrel || ' ' || xtab(i).xphrase);
  end loop;
end;
```

This code produces the following output:

```
PHRASE CANINE
 PT DOG
SYN PUPPY
SYN MUTT
SYN MONGREL
```

Related Topics

OUTPUT_STYLE

SYNonym (SYN) Operator in Chapter 3, "Oracle Text CONTAINS Query Operators"

THES_TT

This procedure finds and returns all top terms of a thesaurus. A top term is defined as any term which has a narrower term but has no broader terms.

This procedure differs from TT in that TT takes in a phrase and finds the top term for that phrase, but THES_TT searches the whole thesaurus and finds all top terms.

Large Thesauri

Because this procedure searches the whole thesaurus, it can take some time on large thesauri. Oracle recommends that you not call this often for such thesauri. Instead, your application should call this once, store the results in a separate table, and use those stored results.

Syntax

```
CTX_THES.THES_TT(restab IN OUT NOCOPY EXP_TAB,
                 tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

restab
Specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about EXP_TAB.

tname
Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This procedure returns all top terms and stores them in restab.

TR

For a given mono-lingual thesaurus, this function returns the foreign language equivalent of a phrase as recorded in the thesaurus.

> **Note:** Foreign language translation is not part of the ISO-2788 or ANSI Z39.19 thesaural standards. The behavior of TR is specific to Oracle Text.

Syntax 1: Table Result

```
CTX_THES.TR(restab IN OUT NOCOPY EXP_TAB,
            phrase IN VARCHAR2,
            lang   IN VARCHAR2 DEFAULT NULL,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT')
```

Syntax 2: String Result

```
CTX_THES.TR(phrase IN VARCHAR2,
            lang   IN VARCHAR2 DEFAULT NULL,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type `EXP_TAB` which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about `EXP_TAB`.

phrase

Specify a phrase to lookup in thesaurus.

lang

Specify the foreign language. Specify `'ALL'` for all translations of `phrase`.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of foreign terms in the form:

```
{ft1}|{ft2}|{ft3} ...
```

Example

Consider a thesaurus `MY_THES` with the following entries for *cat*:

```
cat
  SPANISH: gato
  FRENCH:  chat
  SYN lion
    SPANISH: leon
```

To look up the translation for *cat*, enter the following statements:

```
declare
  trans      varchar2(2000);
  span_trans varchar2(2000);
begin
  trans := ctx_thes.tr('CAT','ALL','MY_THES');
  span_trans := ctx_thes.tr('CAT','SPANISH','MY_THES')
  dbms_output.put_line('the translations for CAT are: '||trans);
  dbms_output.put_line('the Spanish translations for CAT are: '||span_trans);
end;
```

This codes produces the following output:

```
the translations for CAT are: {CAT}|{CHAT}|{GATO}
the Spanish translations for CAT are: {CAT}|{GATO}
```

Related Topics

OUTPUT_STYLE

Translation Term (TR) Operator in Chapter 3, "Oracle Text CONTAINS Query Operators"

TRSYN

For a given mono-lingual thesaurus, this function returns the foreign equivalent of a phrase, synonyms of the phrase, and foreign equivalent of the synonyms as recorded in the specified thesaurus.

> **Note:** Foreign language translation is not part of the ISO-2788 or ANSI Z39.19 thesaural standards. The behavior of TRSYN is specific to Oracle Text.

Syntax 1: Table Result

```
CTX_THES.TRSYN(restab IN OUT NOCOPY EXP_TAB,
               phrase IN VARCHAR2,
               lang   IN VARCHAR2 DEFAULT NULL,
               tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.TRSYN(phrase IN VARCHAR2,
           lang   IN VARCHAR2 DEFAULT NULL,
           tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about EXP_TAB.

phrase

Specify a phrase to lookup in thesaurus.

lang

Specify the foreign language. Specify 'ALL' for all translations of *phrase*.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of foreign terms in the form:

```
{ft1}|{ft2}|{ft3} ...
```

Example

Consider a thesaurus MY_THES with the following entries for *cat*:

```
cat
  SPANISH: gato
  FRENCH:  chat
  SYN lion
    SPANISH: leon
```

To look up the translation and synonyms for *cat*, enter the following statements:

```
declare
  synonyms   varchar2(2000);
  span_syn   varchar2(2000);
begin
  synonyms := ctx_thes.trsyn('CAT','ALL','MY_THES');
  span_syn := ctx_thes.trsyn('CAT','SPANISH','MY_THES')
  dbms_output.put_line('all synonyms for CAT are: '||synonyms);
  dbms_output.put_line('the Spanish synonyms for CAT are: '||span_syn);
end;
```

This codes produces the following output:

```
all synonyms for CAT are: {CAT}|{CHAT}|{GATO}|{LION}|{LEON}
the Spanish synonyms for CAT are: {CAT}|{GATO}|{LION}|{LEON}
```

Related Topics

OUTPUT_STYLE

Translation Term Synonym (TRSYN) Operator in Chapter 3, "Oracle Text CONTAINS Query Operators"

TT

This function returns the top term of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.TT(restab IN OUT NOCOPY EXP_TAB,
            phrase IN VARCHAR2,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.TT(phrase IN VARCHAR2,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN varchar2;
```

restab
Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

> **See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Oracle Text Result Tables" for more information about EXP_TAB.

phrase
Specify a phrase to lookup in thesaurus.

tname
Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns the top term string in the form:

```
{tt}
```

Example

Consider a thesaurus MY_THES with the following broader term entries for *dog*:

```
DOG
   BT1 CANINE
     BT2 MAMMAL
       BT3 VERTEBRATE
         BT4 ANIMAL
```

To look up the top term for *DOG*, execute the following code:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.tt('DOG','MY_THES');
```

```
  dbms_output.put_line('The top term for DOG is: '||terms);
end;
```

This code produces the following output:

```
The top term for dog is: {ANIMAL}
```

Related Topics

OUTPUT_STYLE

Top Term (TT) Operator in Chapter 3, "Oracle Text CONTAINS Query Operators"

UPDATE_TRANSLATION

Use this procedure to update an existing translation.

Syntax

```
CTX_THES.UPDATE_TRANSLATION(tname       in     varchar2,
                            phrase      in     varchar2,
                            language    in     varchar2,
                            translation in     varchar2,
                            new_translation in varchar2);
```

tname
Specify the name of the thesaurus, using no more than 30 characters.

phrase
Specify the phrase in the thesaurus to which to update a translation. The phrase must already exist in the thesaurus or an error is raised.

language
Specify the language of the translation, using no more than 10 characters.

translation
Specify the translated term to update. If no such translation exists, an error is raised.

You can specify `NULL` if there is only one translation for the *phrase.* An error is raised if there is more than one translation for the term in the specified language.

new_translation
Optionally, specify the new form of the translated term.

Example

The following code updates the Spanish translation for *dog*:

```
begin
   ctx_thes.update_translation('my_thes', 'dog', 'SPANISH:', 'PERRO', 'CAN');
end;
```

13

CTX_ULEXER Package

This chapter provides reference information on how to use the CTX_ULEXER PL/SQL package with the user-defined lexer.

CTX_ULEXER declares the following type:

Name	Description
WILDCARD_TAB	Index-by table type that you use to specify the offset of characters to be treated as wildcard characters by the user-defined lexer query procedure.

WILDCARD_TAB

```
TYPE WILDCARD_TAB IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```

Use this index-by table type to specify the offset of those characters in the query word to be treated as wildcard characters by the user-defined lexer query procedure.

Character offset information follows USC-2 codepoint semantics.

14

Oracle Text Utilities

This chapter discusses the utilities shipped with Oracle Text. The following topics are included:

- Thesaurus Loader (ctxload)
- Knowledge Base Extension Compiler (ctxkbtc)
- Lexical Compiler (ctxlc)

Thesaurus Loader (ctxload)

Use `ctxload` to import a thesaurus file into the Oracle Text thesaurus tables.

An import file is an ASCII flat file that contains entries for synonyms, broader terms, narrower terms, or related terms, which can be used to expand queries.

> **See Also:** For examples of import files for thesaurus importing, see "Structure of ctxload Thesaurus Import File" on page C-3 in Appendix C, "Text Loading Examples for Oracle Text"

Text Loading

The `ctxload` program no longer supports the loading of text columns. To load files to a text column in batch mode, Oracle recommends that you use SQL*Loader.

> **See Also:** "SQL*Loader Example" on page C-1 in Appendix C, "Text Loading Examples for Oracle Text"

ctxload Syntax

```
ctxload -user username[/password][@sqlnet_address]
        -name object_name
        -file file_name

       [-thes]
       [-thescase y|n]
       [-thesdump]
       [-log file_name]
       [-trace]
       [-drop]
```

Mandatory Arguments

-user
Specify the user name and password of the user running `ctxload`.

The user name and password can be followed immediately by *@sqlnet_address* to permit logging on to remote databases. The value for *sqlnet_address* is a database connect string. If the TWO_TASK environment variable is set to a remote database, then you do not need to specify a value for *sqlnet_address* to connect to the database.

-name object_name
When you use ctxload to import a thesaurus, use object_name to specify the name of the thesaurus to be imported.

Use object_name to identify the thesaurus in queries that use thesaurus operators.

> **Note:** Thesaurus name must be unique. If the name specified for the thesaurus is identical to an existing thesaurus, then ctxload returns an error and does not overwrite the existing thesaurus.

-file file_name
When ctxload is used to import a thesaurus, use *file_name* to specify the name of the import file that contains the thesaurus entries.

When ctxload is used to export a thesaurus, use *file_name* to specify the name of the export file created by ctxload.

> **Note:** If the name specified for the thesaurus dump file is identical to an existing file, then ctxload overwrites the existing file.

Optional Arguments

-thes
Import a thesaurus. Specify the source file with the -file argument. Specify the name of the thesaurus to be imported with -name.

-thescase y | n
Specify *y* to create a case-sensitive thesaurus with the name specified by -name and populate the thesaurus with entries from the thesaurus import file specified by -file. If -thescase is *y* (the thesaurus is case-sensitive), ctxload enters the terms in the thesaurus exactly as they appear in the import file.

The default for -thescase is *n* (case-insensitive thesaurus).

> **Note:** -thescase is valid for use only with the -thes argument.

-thesdump
Export a thesaurus. Specify the name of the thesaurus to be exported with the -name argument. Specify the destination file with the -file argument.

-log
Specify the name of the log file to which ctxload writes any national-language supported (Globalization Support) messages generated during processing. If you do not specify a log file name, the messages appear on the standard output.

-trace
Enables SQL statement tracing using ALTER SESSION SET SQL_TRACE TRUE. This command captures all processed SQL statements in a trace file, which can be used for

debugging. The location of the trace file is operating-system dependent and can be modified using the USER_DUMP_DEST initialization parameter.

> **See Also:** For more information about SQL trace and the USER_DUMP_DEST initialization parameter, see *Oracle Database Administrator's Guide*

ctxload Examples

This section provides examples for some of the operations that ctxload can perform.

> **See Also:** For more document loading examples, see Appendix C, "Text Loading Examples for Oracle Text"

Thesaurus Import Example

The following example imports a thesaurus named tech_doc from an import file named tech_thesaurus.txt:

```
ctxload -user jsmith/123abc -thes -name tech_doc -file tech_thesaurus.txt
```

Thesaurus Export Example

The following example dumps the contents of a thesaurus named tech_doc into a file named tech_thesaurus.out:

```
ctxload -user jsmith/123abc -thesdump -name tech_doc -file tech_thesaurus.out
```

Knowledge Base Extension Compiler (ctxkbtc)

The knowledge base is the information source that Oracle Text uses to perform theme analysis, such as theme indexing, processing ABOUT queries, and to document theme extraction with the CTX_DOC package. A knowledge base is supplied for English and French.

With the ctxkbtc compiler, you can:

- Extend your knowledge base by compiling one or more thesauri with the Oracle Text knowledge base. The extended information can be application-specific terms and relationships. During theme analysis, the extended portion of the knowledge base overrides any terms and relationships in the knowledge base where there is overlap.
- Create a new user-defined knowledge base by compiling one or more thesauri. In languages other than English and French, this feature can be used to create a language-specific knowledge base.

> **Note:** Only CTXSYS can extend the knowledge base.

> **See Also:** For more information about the knowledge base packaged with Oracle Text, see http://www.oracle.com/technology/products/text/
>
> For more information about the ABOUT operator, see ABOUT operator in Chapter 3, "Oracle Text CONTAINS Query Operators"
>
> For more information about document services, see Chapter 8, "CTX_DOC Package"

Knowledge Base Character Set

Knowledge bases can be in any single-byte character set. Supplied knowledge bases are in WE8ISO8859P1. You can store an extended knowledge base in another character set such as US7ASCII.

ctxkbtc Syntax

```
ctxkbtc -user uname/passwd
[-name thesname1 [thesname2 ... thesname16]]
[-revert]
[-stoplist stoplistname]
[-verbose]
[-log filename]
```

-user
Specify the user name and password for the administrator creating an extended knowledge base. This user must have write permission to the ORACLE_HOME directory.

-name *thesname1 [thesname2 ... thesname16]*
Specify the names of the thesauri (up to 16) to be compiled with the knowledge base to create the extended knowledge base. The thesauri you specify must already be loaded with ctxload with the "-thescase Y" option

-revert
Reverts the extended knowledge base to the default knowledge base provided by Oracle Text.

-stoplist *stoplistname*
Specify the name of the stoplist. Stopwords in the stoplist are added to the knowledge base as useless words that are prevented from becoming themes or contributing to themes. Add stopthemes after running this command using CTX_DLL.ADD_STOPTHEME.

-verbose
Displays all warnings and messages, including non-Globalization Support messages, to the standard output.

-log
Specify the log file for storing all messages. When you specify a log file, no messages are reported to standard out.

ctxkbtc Usage Notes

- Before running ctxkbtc, you must set the NLS_LANG environment variable to match the database character set.
- The user issuing ctxkbtc must have write permission to the ORACLE_HOME, because the program writes files to this directory.
- Before being compiled, each thesaurus must be loaded into Oracle Text case sensitive with the "-thescase Y" option in ctxload.
- Running ctxkbtc twice removes the previous extension.

ctxkbtc Limitations

The ctxkbtc program has the following limitations:

- When upgrading or downgrading your database to a different release, for theme indexing and related features to work correctly, Oracle recommends that you recompile your extended knowledge base in the new environment.
- Before extending the knowledge base, you must terminate all server processes that have invoked any knowledge base-related Text functions during their lifetime.
- There can be only one user extension for each language for each installation. Because a user extension affects all users at the installation, only the `CTXSYS` user can extend the knowledge base.
- In an Oracle RAC environment, the `ORACLE_HOME` can either be shared between multiple nodes, or each node can have its own `ORACLE_HOME`. The following requirements apply:
  - Before using any knowledge base-dependent functionality in any of the Oracle RAC nodes, `ctxkbtc` must be run in every `ORACLE_HOME` in the Oracle RAC environment.
  - When using `ctxkbtc`, the exact same input thesaurus content must be used in every `ORACLE_HOME` in the Oracle RAC environment.

ctxkbtc Constraints on Thesaurus Terms

Terms are case sensitive. If a thesaurus has a term in uppercase, for example, the same term present in lowercase form in a document will not be recognized.

The maximum length of a term is 80 characters.

Disambiguated homographs are not supported.

ctxkbtc Constraints on Thesaurus Relations

The following constraints apply to thesaurus relations:

- BTG and BTP are the same as BT. NTG and NTP are the same as NT.
- Only preferred terms can have a BT, NTs or RTs.
- If a term has no USE relation, it will be treated as its own preferred term.
- If a set of terms are related by SYN relations, only one of them may be a preferred term.
- An existing category cannot be made a top term.
- There can be no cycles in BT and NT relations.
- A term can have at most one preferred term and at most one BT. A term may have any number of NTs.
- An RT of a term cannot be an ancestor or descendent of the term. A preferred term may have any number of RTs up to a maximum of 32.
- The maximum height of a tree is 16 including the top term level.
- When multiple thesauri are being compiled, a top term in one thesaurus should not have a broader term in another thesaurus.

Note: The thesaurus compiler tolerates some violations of the preceding rules. For example, if a term has multiple BTs, then the compiler ignores all but the last one it encounters.

Similarly, BTs between existing knowledge base categories result only in a warning message.

Oracle recommends that you do not set up extended storage bases with violations. Using extended storage bases containing violations can produce undesired results.

Extending the Knowledge Base

Extend the supplied knowledge base by compiling one or more thesauri with the Oracle Text knowledge base. The extended information can be application-specific terms and relationships. During theme analysis, the extended portion of the knowledge base overrides any terms and relationships in the knowledge base where there is overlap.

When extending the knowledge base, Oracle recommends that new terms be linked to one of the categories in the knowledge base for best results in theme proving when appropriate.

See Also: For complete description of the supplied knowledge base, see
http://www.oracle.com/technology/products/text/

If new terms are kept completely disjoint from existing categories, fewer themes from new terms will be proven. The result of this is poorer precision and recall with `ABOUT` queries as well poor quality of gists and theme highlighting.

Link new terms to existing terms by making an existing term the broader term for the new terms.

Example for Extending the Knowledge Base

You purchase a medical thesaurus `medthes` containing a hierarchy of medical terms. The four top terms in the thesaurus are the following:

- Anesthesia and Analgesia
- Anti-Allergic and Respiratory System Agents
- Anti-Inflammatory Agents, Antirheumatic Agents, and Inflammation Mediators
- Antineoplastic and Immunosuppressive Agents

To link these terms to the existing knowledge base, add the following entries to the medical thesaurus to map the new terms to the existing *health and medicine* branch:

```
health and medicine
 NT Anesthesia and Analgesia
 NT Anti-Allergic and Respiratory System Agents
 NT Anti-Inflamammatory Agents, Antirheumatic Agents, and Inflamation Mediators
 NT Antineoplastic and Immunosuppressive Agents
```

Set your globalization support language environment variable to match the database character set. For example, if your database character set is WE8ISO8859P1 and you are using American English, set your `NLS_LANG` as follows:

```
setenv NLS_LANG AMERICAN_AMERICA.WE8ISO8859P1
```

Assuming the medical thesaurus is in a file called med.thes, load the thesaurus as `medthes` with `ctxload` as follows:

```
ctxload -thes -thescase y -name medthes -file med.thes -user ctxsys/ctxsys
```

To link the loaded thesaurus `medthes` to the knowledge base, use `ctxkbtc` as follows:

```
ctxkbtc -user ctxsys/ctxsys -name medthes
```

Adding a Language-Specific Knowledge Base

Extend theme functionality to languages other than English or French by loading your own knowledge base for any single-byte whitespace delimited language, including Spanish.

Theme functionality includes theme indexing, `ABOUT` queries, theme highlighting, and the generation of themes, gists, and theme summaries with the `CTX_DOC` PL/SQL package.

Extend theme functionality by adding a user-defined knowledge base. For example, you can create a Spanish knowledge base from a Spanish thesaurus.

To load your language-specific knowledge base, follow these steps:

1. Load your custom thesaurus using `ctxload`.
2. Set NLS_LANG so that the language portion is the target language. The charset portion must be a single-byte character set.
3. Compile the loaded thesaurus using `ctxkbtc`:

```
ctxkbtc -user ctxsys/ctxsys -name my_lang_thes
```

This command compiles your language-specific knowledge base from the loaded thesaurus. To use this knowledge base for theme analysis during indexing and `ABOUT` queries, specify the `NLS_LANG` language as the `THEME_LANGUAGE` attribute value for the `BASIC_LEXER` preference.

Limitations for Adding a Knowledge Base

The following limitations hold for adding knowledge bases:

- Oracle Text supplies knowledge bases in English and French only. You must provide your own thesaurus for any other language.
- You can only add knowledge bases for languages with single-byte character sets. You cannot create a knowledge base for languages which can be expressed only in multibyte character sets. If the database is a multibyte universal character set, such as UTF-8, the NLS_LANG parameter must still be set to a compatible single-byte character set when compiling the thesaurus.
- Adding a knowledge base works best for whitespace delimited languages.
- You can have at most one knowledge base for each globalization support language.
- Obtaining hierarchical query feedback information such as broader terms, narrower terms and related terms does not work in languages other than English and French. In other languages, the knowledge bases are derived entirely from your thesauri. In such cases, Oracle recommends that you obtain hierarchical information directly from your thesauri.

Order of Precedence for Multiple Thesauri

When multiple thesauri are to be compiled, precedence is determined by the order in which thesauri are listed in the arguments to the compiler, assumed to be *most preferred* first. A user-defined thesaurus always has precedence over the built-in knowledge base.

Size Limits for Extended Knowledge Base

The following table lists the size limits associated with creating and compiling an extended knowledge base.

Table 14–1 Size Limit for the Extended Knowledge Base

Description of Parameter	Limit
Number of RTs (from + to) for each term	32
Number of terms for each single hierarchy (for example, all narrower terms for a given top term)	64000
Number of new terms in an extended knowledge base	1 million
Number of separate thesauri that can be compiled into a user extension to the KB	16

Lexical Compiler (ctxlc)

The Lexical Compiler (`ctxlc`) is a command-line utility that enables you to create your own Chinese and Japanese lexicons (dictionaries). Such a lexicon may either be generated from a user-supplied word list or from the merging of a word list with the system lexicon for that language.

`ctxlc` creates the new lexicon in your current directory. The new lexicon consists of three files, `drold.dat`, `drolk.dat`, and `droli.dat`. To change your system lexicon for Japanese or Chinese, overwrite the system lexicon with these files.

The Lexical Compiler can also generate wordlists from the system lexicons for Japanese and Chinese, enabling you to see their contents. These word lists go to the standard output and thus can be redirected into a file of your choice.

After overwriting the system lexicon, you need to re-create your indexes before querying them.

Syntax of ctxlc

`ctxlc` has the following syntax:

```
ctxlc -ja | -zh [ -n ] -ics character_set -i input_file

ctxlc -ja | -zh -ocs character_set [ > output_file ]
```

Mandatory Arguments

-ja | -zh
Specify the language of the lexicon to modify or create. `-ja` indicates the Japanese lexicon; `-zh` indicates the Chinese lexicon.

-ics *character_set*
Specify the character set of the input file denoted by `-i` *input_file*. *input_file* is the list of words, one word to a line, to use in creating the new lexicon.

-i *input_file*
Specify the file containing words to use in creating a new lexicon.

-ocs *character_set*
Specify the character set of the text file to be output.

Optional Arguments

-n
Specify `-n` to create a new lexicon that consists only of user-supplied words taken from *input_file*. If `-n` is not specified, then the new lexicon consists of a merge of the system lexicon with *input_file*. Also, when `-n` is not selected, a text file called `drolt.dat`, is created in the current directory to enable you to inspect the contents of the merged lexicon without having to enter another `ctxlc` command.

Performance Considerations

You can add up to 1,000,000 new words to a lexicon. However, creating a very large lexicon can reduce performance in indexing and querying. Performance is best when the lexicon character set is UTF-8. There is no performance impact on the Chinese or Japanese V-gram lexers, as they do not use lexicons.

ctxlc Usage Notes

Oracle recommends the following practices with regard to `ctxlc`:

- Save your plain text dictionary file in your environment for emergency use.
- When upgrading or downgrading your database to a different release, recompile your plain text dictionary file in the new environment so that the user lexicon will work correctly.

Example

In this example, you create a new Japanese lexicon from the file `jadict.txt`, a word list that uses the JA16EUC character set. Because you are not specifying `-n`, the new lexicon is the result of merging `jadict.txt` with the system Japanese lexicon. Then replace the existing Japanese lexicon with the new, merged one.

```
% ctxlc -ja -ics JA16EUC -i jadict.txt
```

This creates new files in the current directory:

```
% ls
drold.dat
drolk.dat
droli.dat
drolt.dat
```

The system lexicon files for Japanese and Chinese are named `drold`*xx*`.dat` `drolk`*xx*`.dat`, and `droli`*xx*`.dat`, where *xx* is either `JA` (for Japanese) or `ZH` (for Chinese). Rename the three new files and copy them to the directory containing the system Japanese lexicon.

```
% mv drold.dat droldJA.dat
% mv drolk.dat drolkJA.dat
% mv droli.dat droliJA.dat
% cp *dat $ORACLE_HOME/ctx/data/jalx
```

This replaces the system Japanese lexicon with one that is a merge of the old system lexicon and your wordlist from `jadict.txt`.

You can also use `ctxlc` to get a dump of a system lexicon. This example dumps the Chinese lexicon to a file called `new_chinese_dict.txt` in the current directory:

```
% ctxlc -zh -ocs UTF8 > new_chinese_dict.txt
```

This creates a file, `new_japanese.dict.txt`, using the UTF8 character set, in the current directory.

15

Oracle Text Alternative Spelling

This chapter describes various ways that Oracle Text handles alternative spelling of words. It also documents the alternative spelling conventions that Oracle Text uses in the German, Danish, and Swedish languages.

The following topics are covered:

Overview of Alternative Spelling Features

Some languages have alternative spelling forms for certain words. For example, the German word *Schoen* can also be spelled as *Schön*.

The form of a word is either *original* or *normalized*. The original form of the word is how it appears in the source document. The normalized form is how it is transformed, if it is transformed at all. Depending on the word being indexed and which system preferences are in effect (these are discussed in this chapter), the normalized form of a word may be the same as the original form. Also, the normalized form may comprise more than one spelling. For example, the normalized form of *Schoen* is both *Schoen* and *Schön*.

Oracle Text handles indexing of alternative word forms in the following ways:

- Alternate Spelling—indexing of alternative forms is enabled
- Base-Letter Conversion—accented letters are transformed into non-accented representations
- New German Spelling—reformed German spelling is accepted

Enable these features by specifying the appropriate attribute to the `BASIC_LEXER`. For instance, enable alternate spelling by specifying either `GERMAN`, `DANISH`, or `SWEDISH` for the `ALTERNATE_SPELLING` attribute. As an example, here is how to enable alternate spelling in German:

```
begin
ctx_ddl.create_preference('GERMAN_LEX', 'BASIC_LEXER');
ctx_ddl.set_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING', 'GERMAN');
end;
```

To disable alternate spelling, use the `CTX_DDL.UNSET_ATTRIBUTE` procedure as follows:

```
begin
```

```
ctx_ddl.unset_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING');
end;
```

Oracle Text converts query terms to their normalized forms before lookup. As a result, users can query words with either spelling. If *Schoen* has been indexed as both *Schoen* and *Schön*, a query with *Schön* returns documents containing either form.

Alternate Spelling

When Swedish, German, or Danish has more than one way of spelling a word, Oracle Text normally indexes the word in its original form; that is, as it appears in the source document.

When Alternate Spelling is enabled, Oracle Text indexes words in their normalized form. So, for example, *Schoen* is indexed both as *Schoen* and as *Schön*, and a query on *Schoen* will return documents containing either spelling. (The same is true of a query on *Schön*.)

To enable Alternate Spelling, set the `BASIC_LEXER` attribute `ALTERNATE_SPELLING` to `GERMAN`, `DANISH`, or `SWEDISH`. See "BASIC_LEXER" on page 2-31 for more information.

Base-Letter Conversion

Besides alternative spelling, Oracle Text also handles base-letter conversions. With base-letter conversions enabled, letters with umlauts, acute accents, cedillas, and the like are converted to their basic forms for indexing, so *fiancé* is indexed both as *fiancé* and as *fiance*, and a query of *fiancé* returns documents containing either form.

To enable base-letter conversions, set the `BASIC_LEXER` attribute `BASE_LETTER` to `YES`. See "BASIC_LEXER" on page 2-31 for more information.

When Alternate Spelling is also enabled, Base-Letter Conversion may need to be overridden to prevent unexpected results. See "Overriding Base-Letter Transformations with Alternate Spelling" on page 15-3 for more information.

Generic Versus Language-Specific Base-Letter Conversions

The `BASE_LETTER_TYPE` attribute affects the way base-letter conversions take place. It has two possible values: `GENERIC` or `SPECIFIC`.

The `GENERIC` value is the default and specifies that base letter transformation uses one transformation table that applies to all languages.

The `SPECIFIC` value means that a base-letter transformation that has been specifically defined for your language will be used. This enables you to use accent-sensitive searches for words in your own language, while ignoring accents that are from other languages.

For example, both the `GENERIC` and the Spanish `SPECIFIC` tables will transform *é* into *e*. However, they treat the letter *ñ* distinctly. The `GENERIC` table treats *ñ* as an *n* with an accent (actually, a tilde), and so transforms *ñ* to *n*. The Spanish `SPECIFIC` table treats *ñ* as a separate letter of the alphabet, and thus does not transform it.

New German Spelling

In 1996, new spelling rules for German were approved by representatives from all German-speaking countries. For example, under the spelling reforms, *Potential* becomes *Potenzial*, *Schiffahrt* becomes *Schifffahrt*, and *schneuzen* becomes *schnäuzen*.

When the BASIC_LEXER attribute NEW_GERMAN_SPELLING is set to YES, then a CONTAINS query on a German word that has both new and traditional forms will return documents matching both forms. For example, a query on *Potential* returns documents containing both *Potential* and *Potenzial*. The default setting is NO.

Note: Under reformed German spelling, many words traditionally spelled as one word, such as *soviel*, are now spelled as two (*so viel*). Currently, Oracle Text does not make these conversions, nor conversions from two words to one (for example, *weh tun* to *wehtun*).

The case of the transformed word is determined from the first two characters of the word in the source document; that is, *schiffahrt* becomes *schifffahrt*, *Schiffahrt* becomes *Schifffahrt*, and *SCHIFFAHRT* becomes *SCHIFFFAHRT*.

As many new German spellings include hyphens, it is recommended that users choosing NEW_GERMAN_SPELLING define hyphens as printjoins.

See "BASIC_LEXER" on page 2-31 for more information on setting this attribute.

Overriding Alternative Spelling Features

Even when alternative spelling features have been specified by lexer preference, it is possible to override them. Overriding takes the following form:

- Overriding of base-letter conversion when Alternate Spelling is used, to prevent characters with alternate spelling forms, such as *ü*, *ö*, and *ä*, from also being transformed to the base letter forms.

Overriding Base-Letter Transformations with Alternate Spelling

Transformations caused by turning on alternate_spelling are performed before those of base_letter, which can sometimes cause unexpected results when both are enabled.

When Alternate Spelling is enabled, Oracle Text converts two-letter forms to single-letter forms (for example, *ue* to *ü*), so that words can be searched in both their base and alternate forms. Therefore, with Alternate Spelling enabled, a search for *Schoen* will return documents with both *Schoen* and *Schön*.

However, when Base-letter Transformation is also enabled, the *ü* in *Schlüssel* is transformed into a *u*, producing the non-existent word (in German, anyway) *Schlussel*, and the word is indexed in all three forms.

To prevent this secondary conversion, set the OVERRIDE_BASE_LETTER attribute to TRUE.

OVERRIDE_BASE_LETTER only affects letters with umlauts; accented letters, for example, are still transformed into their base forms.

For more on BASE_LETTER, see "Base-Letter Conversion" on page 15-2.

Alternative Spelling Conventions

The following sections show the alternative spelling substitutions used by Oracle Text.

German Alternate Spelling Conventions

The German alphabet is the English alphabet plus the additional characters: ä ö ü ß. Table 15–1 lists the alternate spelling conventions Oracle Text uses for these characters.

Table 15–1 German Alternate Spelling Conventions

Character	Alternate Spelling Substitution
ä	ae
ü	ue
ö	oe
Ä	AE
Ü	UE
Ö	OE
ß	ss

Danish Alternate Spelling Conventions

The Danish alphabet is the Latin alphabet without the *w*, plus the special characters: *ø æ å*. Table 15–2 lists the alternate spelling conventions Oracle Text uses for these characters.

Table 15–2 Danish Alternate Spelling Conventions

Character	Alternate Spelling Substitution
æ	ae
ø	oe
å	aa
Æ	AE
Ø	OE
Å	AA

Swedish Alternate Spelling Conventions

The Swedish alphabet is the English alphabet without the *w*, plus the additional characters: å ä ö. Table 15–3 lists the alternate spelling conventions Oracle Text uses for these characters.

Table 15–3 Swedish Alternate Spelling Conventions

Character	Alternate Spelling Convention
ä	ae
å	aa
ö	oe
Ä	AE
Å	AA
Ö	OE

A

Oracle Text Result Tables

This appendix describes the structure of the result tables used to store the output generated by the procedures in the CTX_QUERY, CTX_DOC, and CTX_THES packages.

The following topics are discussed in this appendix:

- CTX_QUERY Result Tables
- CTX_DOC Result Tables
- CTX_THES Result Tables and Data Types

CTX_QUERY Result Tables

For the CTX_QUERY procedures that return results, tables for storing the results must be created before the procedure is called. The tables can be named anything, but must include columns with specific names and data types.

This section describes the following types of result tables, and their required columns:

- EXPLAIN Table
- HFEEDBACK Table

EXPLAIN Table

Table A–1 describes the structure of the table to which CTX_QUERY.EXPLAIN writes its results.

Table A–1 EXPLAIN Result Table

Column Name	Datatype	Description
EXPLAIN_ID	VARCHAR2(30)	The value of the explain_id argument specified in the FEEDBACK call.
ID	NUMBER	A number assigned to each node in the query execution tree. The root operation node has ID =1. The nodes are numbered in a top-down, left-first manner as they appear in the parse tree.
PARENT_ID	NUMBER	The ID of the execution step that operates on the output of the ID step. Graphically, this is the parent node in the query execution tree. The root operation node (ID =1) has PARENT_ID = 0.
OPERATION	VARCHAR2(30)	Name of the internal operation performed. Refer to Table A–2 for possible values.

Table A–1 (Cont.) EXPLAIN Result Table

Column Name	Datatype	Description
OPTIONS	VARCHAR2(30)	Characters that describe a variation on the operation described in the OPERATION column. When an OPERATION has more than one OPTIONS associated with it, OPTIONS values are concatenated in the order of processing. See Table A–3 for possible values.
OBJECT_NAME	VARCHAR2(80)	Section name, wildcard term, weight, or threshold value or term to lookup in the index.
POSITION	NUMBER	The order of processing for nodes that all have the same PARENT_ID.The positions are numbered in ascending order starting at 1.
CARDINALITY	NUMBER	Reserved for future use. You should create this column for forward compatibility.

Operation Column Values

Table A–2 shows the possible values for the OPERATION column of the EXPLAIN table.

Table A–2 EXPLAIN Table OPERATION Column

Operation Value	Query Operator	Equivalent Symbol
ABOUT	ABOUT	(none)
ACCUMULATE	ACCUM	,
AND	AND	&
COMPOSITE	(none)	(none)
EQUIVALENCE	EQUIV	=
MINUS	MINUS	-
NEAR	NEAR	;
NOT	NOT	~
NO_HITS	(no hits will result from this query)	
OR	OR	|
PHRASE	(a phrase term)	
SECTION	(section)	
THRESHOLD	>	>
WEIGHT	*	*
WITHIN	within	(none)
WORD	(a single term)	

OPTIONS Column Values

Table A–3 list the possible values for the OPTIONS column of the EXPLAIN table.

Table A–3 EXPLAIN Table OPTIONS Column

Options Value	Description
($)	Stem
(?)	Fuzzy

Table A–3 (Cont.) EXPLAIN Table OPTIONS Column

Options Value	Description
(!)	Soundex
(T)	Order for ordered Near.
(F)	Order for unordered Near.
(n)	A number associated with the max_span parameter for the Near operator.
[9]	Indicates that index_stems is set and query is using token_type 9.

HFEEDBACK Table

Table A–4 describes the table to which CTX_QUERY.HFEEDBACK writes its results.

Table A–4 HFEEDBACK Results Table

Column Name	Datatype	Description
FEEDBACK_ID	VARCHAR2(30)	The value of the feedback_id argument specified in the HFEEDBACK call.
ID	NUMBER	A number assigned to each node in the query execution tree. The root operation node has ID =1. The nodes are numbered in a top-down, left-first manner as they appear in the parse tree.
PARENT_ID	NUMBER	The ID of the execution step that operates on the output of the ID step. Graphically, this is the parent node in the query execution tree. The root operation node (ID =1) has PARENT_ID = 0.
OPERATION	VARCHAR2(30)	Name of the internal operation performed. Refer to Table A–5 for possible values.
OPTIONS	VARCHAR2(30)	Characters that describe a variation on the operation described in the OPERATION column. When an OPERATION has more than one OPTIONS associated with it, OPTIONS values are concatenated in the order of processing. See Table A–6 for possible values.
OBJECT_NAME	VARCHAR2(80)	Section name, wildcard term, weight, threshold value or term to lookup in the index.
POSITION	NUMBER	The order of processing for nodes that all have the same PARENT_ID.The positions are numbered in ascending order starting at 1.
BT_FEEDBACK	CTX_FEEDBACK_TYPE	Stores broader feedback terms. See Table A–7.
PT_FEEDBACK	CTX_FEEDBACK_TYPE	Stores related feedback terms. See Table A–7.
NT_FEEDBACK	CTX_FEEDBACK_TYPE	Stores narrower feedback terms. See Table A–7.

Operation Column Values

Table A–5 shows the possible values for the OPERATION column of the HFEEDBACK table.

Table A–5 HFEEDBACK Results Table OPERATION Column

Operation Value	Query Operator	Equivalent Symbol
ABOUT	ABOUT	(none)
ACCUMULATE	ACCUM	,
AND	AND	&
EQUIVALENCE	EQUIV	=
MINUS	MINUS	-
NEAR	NEAR	;
NOT	NOT	~
OR	OR	|
SECTION	(section)	
TEXT	word or phrase of a text query	
THEME	word or phrase of an ABOUT query	
THRESHOLD	>	>
WEIGHT	*	*
WITHIN	within	(none)

OPTIONS Column Values

Table A–6 list the values for the OPTIONS column of the HFEEDBACK table.

Table A–6 HFEEDBACK Results Table OPTIONS Column

Options Value	Description
(T)	Order for ordered Near.
(F)	Order for unordered Near.
(n)	A number associated with the max_span parameter for the Near operator.

CTX_FEEDBACK_TYPE

The CTX_FEEDBACK_TYPE is a nested table of objects. This datatype is pre-defined in the CTXSYS schema. Use this type to define the columns BT_FEEDBACK, RT_FEEDBACK, and NT_FEEDBACK.

The nested table CTX_FEEDBACK_TYPE holds objects of type CTX_FEEDBACK_ITEM_TYPE, which is also pre-defined in the CTXSYS schema. This object is defined with three members and one method as follows:

Table A–7 CTX_FEEDBACK_ITEM_TYPE

CTX_FEEDBACK_ITEM_TYPE Members and Methods	Type	Description
text	NUMBER	Feedback term.
cardinality	NUMBER	(reserved for future use.)

Table A–7 (Cont.) CTX_FEEDBACK_ITEM_TYPE

CTX_FEEDBACK_ITEM_TYPE Members and Methods	Type	Description
score	NUMBER	(reserved for future use.)

The SQL code that defines these objects is as follows:

```
CREATE OR REPLACE TYPE ctx_feedback_type AS TABLE OF ctx_feedback_item_type;

CREATE OR REPLACE TYPE ctx_feedback_item_type AS OBJECT
(text        VARCHAR2(80),
 cardinality NUMBER,
 score       NUMBER,
 MAP MEMBER FUNCTION rank RETURN REAL,
 PRAGMA RESTRICT_REFERENCES (rank, RNDS, WNDS, RNPS, WNPS)
);

CREATE OR REPLACE TYPE BODY ctx_feedback_item_type AS
   MAP MEMBER FUNCTION rank RETURN REAL IS
   BEGIN
      RETURN score;
   END rank;
END;
```

See Also: For an example of how to select from the HFEEDBACK table and its nested tables, refer to CTX_QUERY.HFEEDBACK in Chapter 10, "CTX_QUERY Package"

CTX_DOC Result Tables

The CTX_DOC procedures return results stored in a table. Before calling a procedure, you must create the table. The tables can be named anything, but must include columns with specific names and data types.

This section describes the following result tables and their required columns:

- Filter Table
- Gist Table
- Highlight Table
- Markup Table
- Theme Table

Filter Table

A filter table stores one row for each filtered document returned by CTX_DOC.FILTER. Filtered documents can be plain text or HTML.

When you call CTX_DOC.FILTER for a document, the document is processed through the filter defined for the text column and the results are stored in the filter table you specify.

Filter tables can be named anything, but must include the following columns, with names and datatypes as specified:

Table A–8 FILTER Result Table

Column Name	Type	Description
QUERY_ID	NUMBER	The identifier for the results generated by a particular call to CTX_DOC.FILTER (only populated when table is used to store results from multiple FILTER calls)
DOCUMENT	CLOB	Text of the document, stored in plain text or HTML.

Gist Table

A Gist table stores one row for each Gist/theme summary generated by CTX_DOC.GIST.

Gist tables can be named anything, but must include the following columns, with names and data types as specified:

Table A–9 Gist Table

Column Name	Type	Description
QUERY_ID	NUMBER	Query ID.
POV	VARCHAR2(80)	Document theme. Case depends of how themes were used in document or represented in the knowledge base. POV has the value of GENERIC for the document GIST.
GIST	CLOB	Text of Gist or theme summary, stored as plain text

Highlight Table

A highlight table stores offset and length information for highlighted terms in a document. This information is generated by CTX_DOC.HIGHLIGHT. Highlighted terms can be the words or phrases that satisfy a word or an ABOUT query.

If a document is formatted, the text is filtered into either plain text or HTML and the offset information is generated for the filtered text. The offset information can be used to highlight query terms for the same document filtered with CTX_DOC.FILTER.

Highlight tables can be named anything, but must include the following columns, with names and datatypes as specified:

Table A–10 Highlight Table

Column Name	Type	Description
QUERY_ID	NUMBER	The identifier for the results generated by a particular call to CTX_DOC.HIGHLIGHT (only populated when table is used to store results from multiple HIGHLIGHT calls)
OFFSET	NUMBER	The position of the highlight in the document, relative to the start of document which has a position of 1.
LENGTH	NUMBER	The length of the highlight.

Markup Table

A markup table stores documents in plain text or HTML format with the query terms in the documents highlighted by markup tags. This information is generated when you call CTX_DOC.MARKUP.

Markup tables can be named anything, but must include the following columns, with names and datatypes as specified:

Table A–11 Markup Table

Column Name	Type	Description
QUERY_ID	NUMBER	The identifier for the results generated by a particular call to CTX_DOC.MARKUP (only populated when table is used to store results from multiple MARKUP calls)
DOCUMENT	CLOB	Marked-up text of the document, stored in plain text or HTML format

Theme Table

A theme table stores one row for each theme generated by CTX_DOC.THEMES. The value stored in the THEME column is either a single theme phrase or a string of parent themes, separated by colons.

Theme tables can be named anything, but must include the following columns, with names and data types as specified:

Table A–12 Theme Table

Column Name	Type	Description
QUERY_ID	NUMBER	Query ID
THEME	VARCHAR2(2000)	Theme phrase or string of parent themes separated by colons (:).
WEIGHT	NUMBER	Weight of theme phrase relative to other theme phrases for the document.

Token Table

A token table stores the text tokens for a document as output by the CTX_DOC.TOKENS procedure. Token tables can be named anything, but must include the following columns, with names and data types as specified.

Table A–13 Token Table

Column Name	Type	Description
QUERY_ID	NUMBER	The identifier for the results generated by a particular call to CTX_DOC.HIGHLIGHT (only populated when table is used to store results from multiple HIGHLIGHT calls)
TOKEN	VARCHAR2(64)	The token string in the text.
OFFSET	NUMBER	The position of the token in the document, relative to the start of document which has a position of 1.
LENGTH	NUMBER	The character length of the token.

CTX_THES Result Tables and Data Types

The CTX_THES expansion functions such as BT, NT, and SYN can return the expansions in a table of type EXP_TAB. Specify the name of your table with the restab argument.

EXP_TAB Table Type

The EXP_TAB table type is a table of rows of type EXP_REC.

The EXP_REC and EXP_TAB types are defined as follows in the CTXSYS schema:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);

type exp_tab is table of exp_rec index by binary_integer;
```

When you call a thesaurus expansion function and specify restab, the system returns the expansion as an EXP_TAB table. Each row in this table is of type EXP_REC and represents a word or phrase in the expansion. Table A–14 describes the fields in EXP_REC:

Table A–14 EXP_TAB Table Type (EXP_REC)

EXP_REC Field	Description
xrel	The xrel field contains the relation of the term to the input term (for example, 'SYN', 'PT', 'RT', and so on). The xrel value is PHRASE when the input term appears in the expansion. For translations, the xrel value is the language.
xlevel	The xlevel field is the level of the relation. This is used mainly when xrel is a hierarchical relation (BT*/NT*). The xlevel field is 0 when xrel is PHRASE. The xlevel field is 2 for translations of synonyms under TRSYN. The xlevel field is 1 for operators that are not hierarchical, such as PT and RT.
xphrase	The xphrase is the related term. This includes a qualifier in parentheses, if one exists for the related term. Compound terms are not de-compounded.

B

Oracle Text Supported Document Formats

This appendix contains a list of the document formats supported by the automatic (AUTO_FILTER) filtering technology. The following topics are covered in this appendix:

- About Document Filtering Technology
- Supported Document Formats

About Document Filtering Technology

The automatic filtering technology in Oracle Text uses Oracle Outside in HTML Export technology. This technology also enables you to convert documents to HTML for document presentation with the CTX_DOC package.

To use automatic filtering for indexing and DML processing, you must specify the AUTO_FILTER object in your filter preference.

To use automatic filtering technology for converting documents to HTML with the CTX_DOC package, you need not use the AUTO_FILTER indexing preference, but you must still set up your environment to use this filtering technology, as described in this appendix.

> **Note:** The underlying technology used by Oracle Text was migrated to Oracle Outside in HTML Export in release 11.1.0.7. See "Formats No Longer Supported in 11.1.0.7" on page B-12 for a list of formats that are no longer supported as a result of this migration. Applications that require support for those formats can use USER_FILTER to plug in third-party filtering technology supporting those formats. See "USER_FILTER" on page 2-24 for more information.

Latest Updates for Patch Releases

The supported platforms and formats listed in this appendix apply for this release. These supported formats are updated for patch releases.

Restrictions on Format Support

Password-protected documents and documents with password-protected content are not supported by the AUTO_FILTER filter.

For other limitations, refer to sections in this chapter concerning specific document types.

Supported Platforms for AUTO_FILTER Document Filtering Technology

Several platforms can take advantage of AUTO_FILTER filter technology.

Supported Platforms

AUTO_FILTER filter technology is supported on the following platforms:

- Windows (x86 32-bit) Windows 2000, Windows 2003, Windows XP, and Windows Vista
- Windows (Itanium 64-bit) Windows .Net Server 2003 Enterprise Edition
- Windows (x86 64-bit) Windows 2003 x64 Standard, Enterprise, and Datacenter Editions (64-bit Extended Systems)
- HP-UX (PA-RISC 64-bit) 11.i
- HP/UX (Itanium 64) 11i
- IBM AIX (32-bit pSeries) 5.1 - 5.3
- iSeries (OS/400 using PASE) V5R2
- Red Hat Linux (x86) Advanced Server 3, 4, and 5
- Red Hat Linux (x86) Red Hat Enterprise Linux (RHEL) 4
- Red Hat Linux (Itanium 64) Advanced Server 3, 4, and 5
- Red Hat Linux (zSeries, 31-bit) Advanced Server 3 and 4
- Red Hat Enterprise Linux AS/ES 3.0, 4.0 and 5.0, x86-64 (AMD64/EM64T)
- Oracle Enterprise Linux 4.0 and 5.0, x86-64 (AMD64/EM64T)
- SuSE Linux (x86) 9, 10, and Enterprise Server 9.0
- SuSE Linux (x86 64-bit) SUSE Enterprise Server (SLES) 9, 10
- SuSE Linux (Itanium 64) Enterprise Server 8
- SuSE Linux (zSeries, 31-bit) 9
- Sun Solaris (SPARC 64-bit) 9.x - 10.x
- Sun Solaris (x86-64-bit) 10x

Note that some of these platforms may not be supported by the Oracle Database.

Filtering on PDF Documents and Security Settings

A PDF document can have different levels of security settings as follows:

Table B–1 AUTO_FILTER Behavior with PDF Security Settings

Security Level	Description	PDF Version	Encryption	AUTO_FILTER Support Level
Level 1	Requires a password for opening the document.	1.2+	40 bit RC4	Not supported.
		1.4+	128 bit RC4	Not supported.
		1.5+	128 bit RC4	Not supported.
		1.6+	128 bit AES	Not supported.
		1.7+	256 bit AES	Not supported.

Table B–1 (Cont.) AUTO_FILTER Behavior with PDF Security Settings

Security Level	Description	PDF Version	Encryption	AUTO_FILTER Support Level
Level 2	Disallows user printing of the document.	1.2+	40 bit RC4	Supported.
		1.4+	128 bit RC4	Supported.
		1.5+	128 bit RC4	Supported.
		1.6+	128 bit AES	Not supported.
		1.7+	256 bit AES	Not supported.
Level 3	Disallows user modification or change of the document.	1.2+	40 bit RC4	Supported.
		1.4+	128 bit RC4	Supported.
		1.5+	128 bit RC4	Supported.
		1.6+	128 bit RC4	Not supported.
		1.7+	256 bit AES	Not supported.
Level 4	Disallows the user from copying or extracting content from the document.	1.2+	40 bit RC4	Supported.
		1.4+	128 bit RC4	Supported.
		1.5+	128 bit RC4	Supported.
		1.6+	128 bit AES	Not supported.
		1.7+	256 bit AES	Not supported.

PDF Filtering Limitations

The following limitations apply when filtering PDF files:

- Multi-byte PDFs are supported, provided the PDF document is created using Character ID-keyed (CID) fonts, predefined CJK CMap files, or `ToUnicode` font encodings, and the document does not contain embedded fonts.
- Embedded fonts in a PDF document are not filtered correctly. They are usually displayed using the question mark (?) replacement character.
- Hyperlinks in a PDF are not active when displayed in a browser or a viewing window.
- Annotations, such as notes, sound, or movies, are not supported.

Environment Variables

No environment variables need to be set by the user.

General Limitations

`AUTO_FILTER` filter technology has the following limitations:

- Any ASCII characters less then 0x20 (decimal 32) are converted to hexadecimal numbers.
- Files larger than 2GB are not handled.

Supported Document Formats

The tables in this section list the document formats that Oracle Text supports for filtering.

Document filtering is used for indexing, DML, and for converting documents to HTML with the CTX_DOC package.

Note: These lists do not represent the complete list of formats that Oracle Text is able to process. The USER_FILTER and PROCEDURE_FILTER enable Oracle Text to process *any* document format, provided an external filter exists that can filter to some textual format like plain-text, HTML, XML, and so forth.

Word Processing and Desktop Publishing Formats

Format	Version
Adobe FrameMaker (MIF)	Versions 3.0, 4.0, 5.0, and 6.0 and Japanese 3.0, 4.0, 5.0, and 6.0 (text only)
ANSI Text	7 and 8 bit
ASCII Text	7 and 8 bit
DEC WPS Plus (DX)	Versions through 3.1
DEC WPS Plus (WPL)	Versions through 4.1
DisplayWrite 2 and 3 (TXT)	All versions
EBCDIC	All versions
Enable	Versions 3.0, 4.0, and 4.5
First Choice	Versions through 3.0
Framework	Version 3.0
Hangul	Versions 97, 2002, and 2005
IBM FFT	All versions
IBM Revisable Form Text	All versions
IBM Writing Assistant	Version 1.01
Just System Ichitaro	Versions 4.x through 6.x, 8.x through 13.x and 2004
JustWrite	Versions through 3.0
Legacy	Versions 1.1
Lotus AMI/AMI Professional	Versions 3.1
Lotus Manuscript	Version 2.0
Lotus Notes DXL	All versions
Lotus Notes NSF	All versions (File ID support only)
Lotus Word Pro (non-Windows)	Versions SmartSuite 97, Millennium, and Millennium 9.6 (text only)
Lotus Word Pro (Windows)	Versions SmartSuite 96, 97, and Millennium and Millennium 9.6
MacWrite II	Version 1.1

Format	Version
MASS11	Versions through 8.0
Microsoft Rich Text Format (RTF)	All versions
Microsoft Word (DOS)	Versions through 6.0
Microsoft Word (Mac)	Versions 4.0 - 2004
Microsoft Word (Windows)	Versions through 2007
Microsoft WordPad	All versions
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (Mac)	Versions through 2.0
Microsoft Works (Windows)	Versions through 4.0
Microsoft Windows Write	Versions through 3.0
MultiMate	Versions through 4.0
Navy DIF	All versions
Nota Bene	Version 3.0
Novell Perfect Works	Version 2.0
Novell/Corel WordPerfect (DOS)	Versions through 6.1
Novell/Corel WordPerfect (Mac)	Versions 1.02 through 3.0
Novell/Corel WordPerfect (Windows)	Versions through 12.0
Office Writer	Versions 4.0 - 6.0
OpenOffice Writer (Windows and UNIX)	OpenOffice version 1.1 and 2.0
PC-File Letter	Versions through 5.0
PC-File+ Letter	Versions through 3.0
PFS:Write	Versions A, B, and C
Professional Write Plus (Windows)	Version 1.0
Q&A (DOS)	Version 2.0
Q&A Write (Windows)	Version 3.0
Samna Word	Versions through Samna Word IV+
Signature	Version 1.0
SmartWare II	Version 1.02
Sprint	Versions through 1.0
StarOffice Writer	Version 5.2 (text only) and 6.x through 8.x
Total Word	Version 1.2
Unicode Text	All versions
UTF-8	All versions
Volkswriter 3 and 4	Versions through 1.0
Wang PC (IWP)	Versions through 2.6
WordMARC	Versions through Composer Plus
WordStar (Windows)	Version 1.0
WordStar 2000 (DOS)	Versions through 3.0

Format	Version
XyWrite	Versions through III Plus

Spreadsheet Formats

Format	Version
Enable	Versions 3.0, 4.0, and 4.5
First Choice	Versions through 3.0
Framework	Version 3.0
Lotus 1-2-3 (DOS & Windows)	Versions through 5.0
Lotus 1-2-3 (OS/2)	Versions through 2.0
Lotus 1-2-3 Charts (DOS & Windows)	Versions through 5.0
Lotus 1-2-3 for SmartSuite	Versions 97 - Millennium 9.6
Lotus Symphony	Versions 1.0, 1.1, and 2.0
Lotus Symphony (Documents, Presentations, Spreadsheets)	Version 1.2
Mac Works	Version 2.0
Microsoft Excel Charts	Versions 2.x - 7.0
Microsoft Excel (Mac)	Versions 3.0 - 4.0, 98, 2001, 2002, 2004, and v.X
Microsoft Excel (Windows)	Versions 2.2 through 2007
Microsoft Multiplan	Version 4.0
Microsoft Works (Windows)	Versions through 4.0
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (Mac)	Versions through 2.0
Mosaic Twin	Version 2.5
Novell Perfect Works	Version 2.0
PFS:Professional Plan	Version 1.0
Quattro Pro (DOS)	Versions through 5.0 (text only)
Quattro Pro (Windows)	Version through 12.0 (text only)
SmartWare II	Version 1.02
StarOffice/OpenOffice Calc (Windows and UNIX)	StarOffice versions 5.2 (text only) through 8.x and OpenOffice version 1.1 and 2.0
SuperCalc 5	Version 4.0
VP Planner 3D	Version 1.0

Presentation Formats

Format	Version
Corel/Novell Presentations	Versions through 12.0
Harvard Graphics (DOS)	Versions 2.x and 3.x

Format	Version
Harvard Graphics (Windows)	Windows versions
Freelance (Windows)	Versions through Millennium 9.6
Freelance (OS/2)	Versions through 2.0
Microsoft PowerPoint (Windows)	Versions 3.0 through 2007
	Versions 97 - 2003 (support for read-only files)
Microsoft PowerPoint (Mac)	Versions 4.0 through v.x
	Versions 97 - 2003 (support for read-only files)
StarOffice/OpenOffice Impress (Windows and UNIX)	StarOffice versions 5.2 (text only) and 6.x through 8.x (full support) and OpenOffice version 1.1 and 2.0 (text only)

Database Formats

Format	Version
Access	Versions through 2.0
dBASE	Versions through 5.0
DataEase	Version 4.x
dBXL	Version 1.3
Enable	Versions 3.0, 4.0, and 4.5
First Choice	Versions through 3.0
FoxBase	Version 2.1
Framework	Version 3.0
Microsoft Works (Windows)	Versions through 4.0
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (Mac)	Versions through 2.0
Paradox (DOS)	Versions through 4.0
Paradox (Windows)	Versions through 1.0
Personal R:BASE	Version 1.0
R:BASE 5000	Versions through 3.1
R:BASE System V	Version 1.0
Reflex	Version 2.0
Q & A	Versions through 2.0
SmartWare II	Version 1.02

Archive File Format

When filtering an archive file, all the contents of the files inside the archive will be exported to a single output file. This will also include the contents of all subfolders and files inside the archive file.

Table B–2 lists the archive formats that Oracle supports.

Table B–2 Supported Archive File Formats

Format	Version
GZIP	
Microsoft Binder	Versions 7.0 - 97 (conversion of files contained in the Binder File is supported only on Windows)
UUEncode	
UNIX Compress	
UNIX Tar	
ZIP	PKWARE versions through 2.04g
LZA Self-Extracting Compress	
LZH Compress	

Email Formats

Format	Version
Microsoft Outlook Folder (PST)	Microsoft Outlook Folder and Microsoft Outlook Offline Folder files versions 97, 98, 2000, 2002, 2003, and 2007
Microsoft Outlook Message (MSG)	Microsoft Outlook Message and Microsoft Outlook Form Template versions 97, 98, 2000, 2002, 2003, and 2007
MIME	MIME-encoded mail messages.

MIME Support Notes

The following formats are supported:

- MIME formats
  - EML
  - MHT (Web Archive)
  - NWS (Newsgroup single-part and multi-part)
  - Simple Text Mail (defined in RFC 2822)
- TNEF format
- MIME encodings, including
  - base64 (defined in RFC 1521)
  - binary (defined in RFC 1521)
  - binhex (defined in RFC 1741)
  - btoa
  - quoted-printable (defined in RFC 1521)
  - utf-7 (defined in RFC 2152)
  - uue
  - xxe

  - yenc

In addition, the body of a message can be encoded in several ways. The following encodings are supported:

- HTML
- RTF
- TNEF
- Text/enriched (defined in RFC 1523)
- Text/richtext (defined in RFC1341)
- Embedded mail message (defined in RFC 822) - this is handled as a link to a new message

The attachments of a MIME message can be stored in many formats. Oracle Corporation processes all attachment types that its technology supports.

Other Formats

Format	Version
ASF (subformats: WMA/SMV/DVR-MS)	Metadata extraction only
Executable (EXE, DLL)	-
HTML	Versions through 3.0, with some limitations
IBM Lotus Notes DXL	All versions
IBM Lotus Notes NSF	All versions (File ID support only)
ISO Base Media (subformats: Quicktime/MPEG-4/MPEG-7)	Media extraction only
MacroMedia Flash	Macromedia Flash 6.x, MacroMedia Flash 7.x, and MacroMedia Flash Lite (text only)
Microsoft Office	2007 (support for SmartArt created using SP2 MSO)
Microsoft Office 2008 for Mac	(Word, PowerPoint, Excel)
Microsoft Project	Versions 98 - 2003 (text only)
Microsoft Project	Version 2007 (File ID support only)
Microsoft Publisher	2003/2007 (File ID support only)
Microsoft XPS	Metadata extraction only
MP3	ID3 information
RIFF (subformats: WAV/AVI)	Metadata extraction only
RPIX	File ID support only
vCard, vCalendar	Version 2.1
Windows Executable	-
WML	Version 5.2
XML	Text only
Yahoo Instant	-

Graphic Formats

Table B–3 lists the graphic formats that the AUTO_FILTER filter recognizes. This means that indexing a text column that contains any of these formats produces no error. As such, it is safe for the column to contain any of these formats.

Formats are categorized as either *embedded graphics* or *standalone graphics*. Embedded graphics are inserted or referenced within a document.

Note: The AUTO_FILTER filter cannot extract textual information from graphics.

Table B–3 Supported Graphics Formats for AUTO_FILTER Filter

Graphics Format	Version
Adobe Photoshop (PSD)	Version 4.0
Windows Icon Cursor (ICO)	no specific version
Adobe Photoshop (PSD)	Version 4.0
Adobe Illustrator	Versions 7.0 and 9.0
Adobe FrameMaker graphics (FMV)	Vector/raster through 5.0
Adobe Acrobat (PDF)	Versions 1.0, 2.1, 3.0, 4.0, 5.0, 6.0, and 7.0 (including Japanese PDF)
	Versions 1.6 and 1.7 (PDF packages and portfolios are not supported)
Ami Draw (SDW)	Ami Draw
AutoCAD Interchange and Native Drawing formats (DXF and DWG)	AutoCAD Drawing Versions 2.5 - 2.6, 9.0-14.0, 2000i and 2002
AutoShade Rendering (RND)	Version 2.0
Binary Group 3 Fax	All versions
Bitmap (BMP, RLE, ICO, CUR, OS/2 DIB, and WARP)	All versions
CALS Raster (GP4)	Type I and Type II
Corel Clipart format (CMX)	Versions 5 through 6
Corel Draw (CDR)	Versions 3.x - 8.x
Corel Draw (CDR with TIFF header)	Versions 2.x - 9.x
Computer Graphics Metafile (CGM)	ANSI, CALS NIST version 3.0
Encapsulated PostScript (EPS)	TIFF header only
GEM Paint (IMG)	All versions
Graphics Environment Mgr (GEM)	Bitmap and vector
Graphics Interchange Format (GIF)	All versions
Hewlett Packard Graphics Language (HPGL)	Version 2.0
IBM Graphics Data Format (GDF)	Version 1.0
IBM Picture Interchange Format (PIF)	Version 1.0
Initial Graphics Exchange Spec (IGES)	Version 5.1

Table B–3 (Cont.) Supported Graphics Formats for AUTO_FILTER Filter

Graphics Format	Version
JBIG2	JBIG2 graphic embeddings in PDF files
JFIF (JPEG not in TIFF format)	All versions
JPEG (including EXIF)	All versions
Kodak Flash Pix (FPX)	All versions
Kodak Photo CD (PCD)	Version 1.0
Lotus PIC	All versions
Lotus Snapshot	All versions
Macintosh PIC1 and PICT2	Bitmap only
MacPaint (PNTG)	All versions
Micrografx Draw (DRW)	Versions through 4.0
Micrografx Designer (DRW)	Versions through 3.1
Micrografx Designer (DFS)	Windows 95, version 6.0
Novell PerfectWorks (Draw)	Version 2.0
OS/2 PM Metafile (MET)	Version 3.0
Paint Shop Pro 6 (PSP)	Windows only, versions 5.0 - 6.0
PC Paintbrush (PCX and DCX)	All versions
Portable Bitmap (PBM)	All versions
Portable Graymap (PGM)	No specific version
Portable Network Graphics (PNG)	Version 1.0
Portable Pixmap (PPM)	No specific version
Postscript (PS)	Levels 1-2
Progressive JPEG	No specific version
Sun Raster (SRS)	No specific version
StarOffice/OpenOffice Draw for Windows and UNIX	StarOffice versions 5.2 (text only) through 8.x and OpenOffice version 1.1 and 2.0
TIFF	Versions through 6
TIFF CCITT Group 3 and 4	Versions through 6
Truevision TGA (TARGA)	Version 2
Visio (preview)	Version 4
Visio	Versions 5, 2000, 2002, and 2003
Visio 2007	File ID support only
WBMP	No specific version
Windows Enhanced Metafile (EMF)	No specific version
Windows Metafile (WMF)	No specific version
WordPerfect Graphics (WPG and WPG2)	Versions through 2.0
X-Windows Bitmap (XBM)	x10 compatible
X-Windows Dump (XWD)	x10 compatible

Table B–3 (Cont.) Supported Graphics Formats for AUTO_FILTER Filter

Graphics Format	Version
X-Windows Pixmap (XPM)	x10 compatible

Graphics Formats Limitations

The AUTO_FILTER filter supports AutoCAD on IBM AIX with the following limitations:

- Oracle Database release 11.1.0.7 and release 11.2.0.1 supports AutoCAD files up to AutoCAD 2002.
- Support for AutoCAD versions later than 2002 on IBM AIX is available with Oracle release 11.2.0.2.

Formats No Longer Supported in 11.1.0.7

Certain document formats are not supported if you upgrade from release 11.1.0.6 to 11.1.0.7. This is because Oracle Text filtering technology has been migrated to Oracle Outside in HTML Export technology. To filter these unsupported formats, you can plug in a third party filtering technology using USER_FILTER. See "USER_FILTER" on page 2-24 for more information.

Table B–4 lists the formats supported in release 11.1.0.6, but not in 11.1.0.7.

Table B–4 Formats Supported in Release 11.1.0.6 and not in 11.1.0.7

Format	Versions
Word Processing Formats	
Applix Words (AW)	3.11, 4.0, 4.1, 4.2, 4.3, 4.4
JustSystems Ichitaro (JTD)	2005
Folio Flat File (FFF)	3.1
Fujitsu Oasys (OA2)	7
Lotus Word Pro (LWP)	9.7, 9.8
WordPerfect for Linux	All versions
Desktop Publishing Formats	
Adobe Framemaker (MIF)	7
Spreadsheet Formats	
Applix Spreadsheets (AS)	4.2, 4.3, 4.4
Lotus 1-2-3 (123)	Millennium Edition R9, 9.8
Microsoft Works Spreadsheet (DOS)	3.4
Microsoft Works Spreadsheet (Mac)	3.4
Comma-Separated Values (SCV)	N/A
Presentation Formats	
Applix Presents (AG)	4.0, 4.2, 4.3, 4.4

Table B–4 (Cont.) Formats Supported in Release 11.1.0.6 and not in 11.1.0.7

Format	Versions
Lotus Freelance Graphics (PRE)	Millennium Edition R9, 9.8
Microsoft Visio XML Format	2003
Graphic Formats	
SGI RGB Image (RGB)	No specific version
Windows Animated Cursor (ANI)	No specific version
WordPerfect Graphics 2 (WPG2)	7
Microsoft Office Drawing (MSO)	No specific version
Windows Icon Cursor (ICO)	No specific version

C

Text Loading Examples for Oracle Text

This appendix provides examples of how to load text into a text column, and the structure of `ctxload` import files. This appendix contains these topics:

- SQL INSERT Example
- SQL*Loader Example
- Structure of ctxload Thesaurus Import File

SQL INSERT Example

A simple way to populate a text table is to create a table with two columns, `id` and `text`, using `CREATE TABLE` and then use the `INSERT` statement to load the data. This example makes the `id` column the primary key, which is optional. The `text` column is `VARCHAR2`:

```
create table docs (id number primary key, text varchar2(80));
```

To populate the `text` column, use the `INSERT` statement as follows:

```
insert into docs values(1, 'this is the text of the first document');
insert into docs values(12, 'this is the text of the second document');
```

SQL*Loader Example

The following example shows how to use SQL*Loader to load mixed format documents from the operating system to a `BLOB` column. The example has two steps:

- Create the table
- Enter the SQL*Loader command that reads control file and loads data into table

> **See Also:** For a complete discussion on using SQL*Loader, see *Oracle Database Utilities*

Creating the Table

This example loads a table `articles_formatted` created as follows:

```
CREATE TABLE articles_formatted (
  ARTICLE_ID   NUMBER PRIMARY KEY ,
  AUTHOR       VARCHAR2(30),
  FORMAT       VARCHAR2(30),
  PUB_DATE     DATE,
  TITLE        VARCHAR2(256),
  TEXT         BLOB
```

```
);
```

The article_id column is the primary key. Documents are loaded in the text column, which is of type BLOB.

Issuing the SQL*Loader Command

The following command starts the loader, which reads the control file LOADER1.DAT:

```
sqlldr userid=demo/demo control=loader1.dat log=loader.log
```

Example Control File: loader1.dat

This SQL*Loader control file defines the columns to be loaded and instructs the loader to load the data line by line from loader2.dat into the articles_formatted table. Each line in loader2.dat holds a comma separated list of fields to be loaded.

```
-- load file example
load data
INFILE 'loader2.dat'
INTO TABLE articles_formatted
APPEND
FIELDS TERMINATED BY ','
(article_id SEQUENCE (MAX,1),
 author CHAR(30),
 format,
 pub_date SYSDATE,
 title,
 ext_fname FILLER CHAR(80),
 text LOBFILE(ext_fname) TERMINATED BY EOF)
```

This control file instructs the loader to load data from loader2.dat to the articles_formatted table in the following way:

1. The ordinal position of the line describing the document fields in loader2.dat is written to the article_id column.
2. The first field on the line is written to author column.
3. The second field on the line is written to the format column.
4. The current date given by SYSDATE is written to the pub_date column.
5. The title of the document, which is the third field on the line, is written to the title column.
6. The name of each document to be loaded is read into the ext_fname temporary variable, and the actual document is loaded in the text BLOB column:

Example Data File: loader2.dat

This file contains the data to be loaded into each row of the table, articles_formatted.

Each line contains a comma separated list of the fields to be loaded in articles_formatted. The last field of every line names the file to be loaded in to the text column:

```
Ben Kanobi, plaintext,Kawasaki news article,../sample_docs/kawasaki.txt,
Joe Bloggs, plaintext,Java plug-in,../sample_docs/javaplugin.txt,
John Hancock, plaintext,Declaration of Independence,../sample_docs/indep.txt,
M. S. Developer, Word7,Newsletter example,../sample_docs/newsletter.doc,
M. S. Developer, Word7,Resume example,../sample_docs/resume.doc,
```

```
X. L. Developer, Excel7,Common example,../sample_docs/common.xls,
X. L. Developer, Excel7,Complex example,../sample_docs/solvsamp.xls,
Pow R. Point, Powerpoint7,Generic presentation,../sample_docs/generic.ppt,
Pow R. Point, Powerpoint7,Meeting presentation,../sample_docs/meeting.ppt,
Java Man, PDF,Java Beans paper,../sample_docs/j_bean.pdf,
Java Man, PDF,Java on the server paper,../sample_docs/j_svr.pdf,
Ora Webmaster, HTML,Oracle home page,../sample_docs/oramnu97.html,
Ora Webmaster, HTML,Oracle Company Overview,../sample_docs/oraoverview.html,
John Constable, GIF,Laurence J. Ellison : portrait,../sample_docs/larry.gif,
Alan Greenspan, GIF,Oracle revenues : Graph,../sample_docs/oragraph97.gif,
Giorgio Armani, GIF,Oracle Revenues : Trend,../sample_docs/oratrend.gif,
```

Structure of ctxload Thesaurus Import File

The import file must use the following format for entries in the thesaurus:

```
phrase
 BT broader_term
 NT narrower_term1
 NT narrower_term2
. . .
 NT narrower_termN

 BTG broader_term
 NTG narrower_term1
 NTG narrower_term2
. . .
 NTG narrower_termN

 BTP broader_term
 NTP narrower_term1
 NTP narrower_term2
. . .
 NTP narrower_termN

 BTI broader_term
 NTI narrower_term1
 NTI narrower_term2
. . .
 NTI narrower_termN

 SYN synonym1
 SYN synonym2
. . .
 SYN synonymN

 USE synonym1  or  SEE synonym1 or  PT  synonym1

 RT related_term1
 RT related_term2
. . .
 RT related_termN

 SN text

 language_key: term
```

phrase
is a word or phrase that is defined as having synonyms, broader terms, narrower terms, or related terms.

In compliance with ISO-2788 standards, a TT marker can be placed before a phrase to indicate that the phrase is the top term in a hierarchy; however, the TT marker is not required. In fact, ctxload ignores TT markers during import.

A top term is identified as any phrase that does not have a broader term (BT, BTG, BTP, or BTI).

Note: The thesaurus query operators (`SYN`, `PT`, `BT`, `BTG`, `BTP`, `BTI`, `NT`, `NTG`, `NTP`, `NTI`, and `RT`) are reserved words and, thus, cannot be used as phrases in thesaurus entries.

BT, BTG, BTP, BTI broader_termN
are the markers that indicate broader_termN is a broader (generic | partitive | instance) term for phrase.

broader_termN is a word or phrase that conceptually provides a more general description or category for phrase. For example, the word *elephant* could have a broader term of *land mammal*.

NT, NTG, NTP, NTI narrower_termN
are the markers that indicate narrower_termN is a narrower (generic | partitive | instance) term for phrase.

If phrase does not have a broader (generic | partitive | instance) term, but has one or more narrower (generic | partitive | instance) terms, phrase is created as a top term in the respective hierarchy (in an Oracle Text thesaurus, the BT/NT, BTG/NTG, BTP/NTP, and BTI/NTI hierarchies are separate structures).

narrower_termN is a word or phrase that conceptually provides a more specific description for phrase. For example, the word *elephant* could have a narrower terms of *indian elephant* and *african elephant*.

SYN synonymN
is a marker that indicates phrase and synonymN are synonyms within a synonym ring.

synonymN is a word or phrase that has the same meaning for phrase. For example, the word *dog* could have a synonym of *canine*.

Note: Synonym rings are not defined explicitly in Oracle Text thesauri. They are created by the transitive nature of synonyms.

USE SEE PT synonym1
are markers that indicate phrase and synonym1 are synonyms within a synonym ring (similar to SYN).

The markers USE, SEE or PT also indicate synonym1 is the preferred term for the synonym ring. Any of these markers can be used to define the preferred term for a synonym ring.

Note: If the user-defined thesaurus is to be used for compiling the Knowledge Base, then you must specify the preferred term when a synonym ring is declared. Use one of the keywords USE, SEE, or PT to specify which synonym to use when reporting query matches. Only one term can be a preferred term.

Not using one of these keywords may result in the failure to return results defined by a word's synonym. When compiling two or more thesauri that declare elements of the same synonym ring, the preferred term must be the same in both files, which ensures that only one word is defined as the preferred word in a synonym ring.

RT related_termN
is the marker that indicates related_termN is a related term for phrase.

related_termN is a word or phrase that has a meaning related to, but not necessarily synonymous with phrase. For example, the word *dog* could have a related term of *wolf*.

Note: Related terms are not transitive. If a phrase has two or more related terms, the terms are related only to the parent phrase and not to each other.

SN text
is the marker that indicates the following text is a scope note (for example, comment) for the preceding entry.

language_key term
term is the translation of phrase into the language specified by language_key.

Alternate Hierarchy Structure

In compliance with thesauri standards, the load file supports formatting hierarchies (BT/NT, BTG/NTG, BTP, NTP, BTI/NTI) by indenting the terms under the top term and using NT (or NTG, NTP, NTI) markers that include the level for the term:

```
phrase
   NT1 narrower_term1
      NT2 narrower_term1.1
      NT2 narrower_term1.2
          NT3 narrower_term1.2.1
          NT3 narrower_term1.2.2
   NT1 narrower_term2
   . . .
   NT1 narrower_termN
```

Using this method, the entire branch for a top term can be represented hierarchically in the load file.

Usage Notes for Terms in Import Files

The following conditions apply to the structure of the entries in the import file:

- Each entry (phrase, BT, NT, or SYN) must be on a single line followed by a newline character.
- Entries can consist of a single word or phrases.

- The maximum length of an entry (phrase, BT, NT, or SYN) is 255 bytes, not including the BT, NT, and SYN markers or the newline characters.
- Entries cannot contain parentheses or plus signs.
- Each line of the file that starts with a relationship (BT, NT, and so on) must begin with at least one space.
- A phrase can occur more than once in the file.
- Each phrase can have one or more narrower term entries (NT, NTG, NTP), broader term entries (BT, BTG, BTP), synonym entries, and related term entries.
- Each broader term, narrower term, synonym, and preferred term entry must start with the appropriate marker and the markers must be in capital letters.
- The broader terms, narrower terms, and synonyms for a phrase can be in any order.
- Homographs must be followed by parenthetical disambiguators everywhere they are used.

  For example: `cranes (birds), cranes (lifting equipment)`
- Compound terms are signified by a plus sign between each factor (for example, buildings + construction).
- Compound terms are allowed only as synonyms or preferred terms for other terms, never as terms by themselves, or in hierarchical relations.
- Terms can be followed by a scope note (SN), total maximum length of 2000 bytes, on subsequent lines.
- Multi-line scope notes are allowed, but require an SN marker on each line of the note.

Example of Incorrect SN usage:

```
VIEW CAMERAS
 SN Cameras with through-the lens focusing and a
range of movements of the lens plane relative to
the film plane
```

Example of Correct SN usage:

```
VIEW CAMERAS
 SN Cameras with through-the lens focusing and a
 SN range of movements of the lens plane relative
 SN to the film plane
```

- Multi-word terms cannot start with reserved words (for example, *use* is a reserved word, so *use other door* is not an allowed term; however, *use* is an allowed term).

Usage Notes for Relationships in Import Files

The following conditions apply to the relationships defined for the entries in the import file:

- related term entries must follow a phrase or another related term entry
- related term entries start with one or more spaces, the RT marker, followed by white space, then the related term on the same line
- multiple related terms require multiple RT markers

  Example of incorrect RT usage:

```
MOVING PICTURE CAMERAS
 RT CINE CAMERAS
TELEVISION CAMERAS
```

Example of correct RT usage:

```
MOVING PICTURE CAMERAS
 RT CINE CAMERAS
 RT TELEVISION CAMERAS
```

- Terms are allowed to have multiple broader terms, narrower terms, and related terms

Examples of Import Files

This section provides three examples of correctly formatted thesaurus import files.

Example 1 (Flat Structure)

```
cat
 SYN feline
 NT domestic cat
 NT wild cat
 BT mammal
mammal
 BT animal
domestic cat
 NT Persian cat
 NT Siamese cat
wild cat
 NT tiger
tiger
 NT Bengal tiger
dog
 BT mammal
 NT domestic dog
 NT wild dog
 SYN canine
domestic dog
 NT German Shepard
wild dog
 NT Dingo
```

Example 2 (Hierarchical)

```
animal
   NT1 mammal
        NT2 cat
             NT3 domestic cat
                  NT4 Persian cat
                  NT4 Siamese cat
             NT3 wild cat
                  NT4 tiger
                       NT5 Bengal tiger
        NT2 dog
             NT3 domestic dog
                  NT4 German Shepard
             NT3 wild dog
                  NT4 Dingo
cat
 SYN feline
```

```
dog
 SYN canine
```

Example 3

```
35MM CAMERAS
 BT MINIATURE CAMERAS
CAMERAS
 BT OPTICAL EQUIPMENT
 NT MOVING PICTURE CAMERAS
 NT STEREO CAMERAS
LAND CAMERAS
 USE VIEW CAMERAS
VIEW CAMERAS
 SN Cameras with through-the lens focusing and a range of
 SN movements of the lens plane relative to the film plane
 UF LAND CAMERAS
 BT STILL CAMERAS
```

D

Oracle Text Multilingual Features

This Appendix describes the multilingual features of Oracle Text. The following topics are discussed:

Introduction

This appendix summarizes the main multilingual features for Oracle Text.

For a complete list of Oracle Globalization Support languages and character set support, refer to the *Oracle Database Globalization Support Guide*.

Indexing

The following sections describe the multilingual indexing features:

Multilingual Features for Text Index Types

The following sections describes the supported multilingual features for the Oracle Text index types.

> **See Also:** "Lexer Types" on page D-2 for a description of available lexers

CONTEXT Index Type

The `CONTEXT` index type fully supports multilingual features, including use of the language and character set columns. The following lexers are supported:

- MULTI_LEXER
- USER_LEXER
- WORLD_LEXER

CONTEXT also supports use of all Chinese, Japanese, and Korean language lexers as follows:

- CHINESE_LEXER
- CHINESE_VGRAM_LEXER
- JAPANESE_LEXER
- JAPANESE_VGRAM_LEXER
- KOREAN_MORPH_LEXER

CTXCAT Index Type

CTXCAT supports the multilingual features of the BASIC_LEXER with the exception of indexing themes, and supports the following additional lexers:

- USER_LEXER
- WORLD_LEXER

CTXCAT also supports the following lexers:

- CHINESE_LEXER
- CHINESE_VGRAM_LEXER
- JAPANESE_LEXER
- JAPANESE_VGRAM_LEXER
- KOREAN_MORPH_LEXER

CTXRULE Index Type

The CTXRULE index type supports the multilingual features of the BASIC_LEXER including ABOUT and STEM operators. It also supports Japanese, Chinese, and Korean (when used with the SVM_CLASSIFIER).

Lexer Types

Oracle Text supports the indexing of different languages by enabling you to choose a lexer in the indexing process. The lexer you employ determines the languages you can index. Table D–1 describes the supported lexers.

Table D–1 Oracle Text Lexer Types

Lexer	Supported Languages
BASIC_LEXER	English and most western European languages that use white space delimited words.
MULTI_LEXER	Lexer for indexing tables containing documents of different languages such as English, German, and Japanese.
CHINESE_VGRAM	Lexer for extracting tokens from Chinese text.

Table D–1 (Cont.) Oracle Text Lexer Types

Lexer	Supported Languages
CHINESE_LEXER	Lexer for extracting tokens from Chinese text. This lexer offers the following benefits over the CHINESE_VGRAM lexer: ■ Generates a smaller index ■ Better query response time ■ Generates real world tokens resulting in better query precision ■ Supports stop words
JAPANESE_VGRAM	Lexer for extracting tokens from Japanese text.
JAPANESE_LEXER	Lexer for extracting tokens from Japanese text. This lexer offers the following advantages over the JAPANESE_VGRAM lexer: ■ Generates smaller index ■ Better query response time ■ Generates real world tokens resulting in better precision
KOREAN_MORPH_LEXER	Lexer for extracting tokens from Korean text.
USER_LEXER	Lexer you create to index a particular language.
WORLD_LEXER	Lexer for indexing tables containing documents of different languages; autodetects languages in a document

Basic Lexer Features

The following features are supported with the BASIC_LEXER preference. Enable these features with attributes of the BASIC_LEXER. Features such as alternate spelling, composite, and base letter can be enabled together for better search results.

Theme Indexing

Enables the indexing and subsequent querying of document concepts with the ABOUT operator with CONTEXT index types. These concepts are derived from the Oracle Text knowledge base. This feature is supported for English and French.

This feature is not supported with CTXCAT index types.

Alternate Spelling

This feature enables you to search on alternate spellings of words. For example, with alternate spelling enabled in German, a query on *gross* returns documents that contain *groß* and *gross*.

This feature is supported in German, Danish, and Swedish.

Additionally, German can be indexed according to both traditional and reformed spelling conventions.

> **See Also:** "Alternate Spelling" on page 15-2 and "New German Spelling" on page 15-2.

Base Letter Conversion

This feature enables you to query words with or without diacritical marks such as tildes, accents, and umlauts. For example, with a Spanish base-letter index, a query of *energia* matches documents containing both *energía* and *energia*.

This feature is supported for English and all other supported whitespace delimited languages. In English and French, you can use the basic lexer to enable theme indexing.

See Also: "Base-Letter Conversion" on page 15-2

Composite

This feature enables you to search on words that contain the specified term as a sub-composite. You must use the stem ($) operator. This feature is supported for German and Dutch.

For example, in German, a query of *$register* finds documents that contain *Bruttoregistertonne* and *Registertonne*.

Index stems

This feature enables you to specify a stemmer for stem indexing. Tokens are stemmed to a single base form at index time in addition to the normal forms. Specifying index stems enables better query performance for stem queries, for example *$computed*.

This feature is supported for English, Dutch, French, German, Italian, and Spanish.

Multi Lexer Features

The `MULTI_LEXER` lexer enables you to index a column that contains documents of different languages. During indexing Oracle Text examines the language column and switches in the language-specific lexer to process the document. Define the lexer preferences for each language before indexing.

The multi lexer enables you to set different preferences for languages. For example, you can have `composite` set to `TRUE` for German documents and `composite` set to `FALSE` for Dutch documents.

World Lexer Features

Like `MULTI_LEXER`, the `WORLD_LEXER` lexer enables you to index documents that contain different languages. It automatically detects the languages of a document and, therefore, does not require you to create a language column in the base table.

`WORLD_LEXER` processes all database character sets and supports the Unicode 5.0 standard. For `WORLD_LEXER` to be effective with documents that use multiple languages, AL32UTF-8 or UTF8 Oracle character set encoding must be specified. This includes supplementary, or "surrogate-pair," characters.

Table D–2 and Table D–3 show the languages supported by `WORLD_LEXER`. This list may change as the Unicode standard changes, and in any case should not be considered exhaustive. (Languages are grouped by Unicode writing system, not by natural language groupings.)

Table D–2 Languages Supported by the World Lexer (Space-separated)

Language Group	Languages Include
Arabic	Arabic, Farsi, Kurdish, Pashto, Sindhi, Urdu
Armenian	Armenian
Bengali	Assamese, Bengali
Bopomofo	Hakka Chinese, Minnan Chinese

Table D–2 (Cont.) Languages Supported by the World Lexer (Space-separated)

Language Group	Languages Include
Cyrillic	Over 50 languages, including Belorussian, Bulgarian, Macedonian, Moldavian, Russian, Serbian, Serbo-Croatian, Ukrainian
Devenagari	Bhojpuri, Bihari, Hindi, Kashmiri, Marathi, Nepali, Pali, Sanskrit
Ethiopic	Amharic, Ge'ez, Tigrinya, Tigre
Georgian	Georgian
Greek	Greek
Gujarati	Gujarati, Kacchi
Gurmukhi	{Punjabi
Hebrew	Hebrew, Ladino, Yiddish
Kaganga	Redjang
Kannada	Kanarese, Kannada
Korean	Korean, Hanja Hangul
Latin	Afrikaans, Albanian, Basque, Breton, Catalan, Croatian, Czech, Danish, Dutch, English, Esperanto, Estonian, Faeroese, Fijian, Finnish, Flemish, French, Frisian, German, Hawaiian, Hungarian, Icelandic, Indonesian, Irish, Italian, Lappish, Classic Latin, Latvian, Lithuanian, Malay, Maltese, Pinyin Mandarin, Maori, Norwegian, Polish, Portuguese, Provencal, Romanian, Rumanian, Samoan, Scottish Gaelic, Slovak, Slovene, Slovenian, Sorbian, Spanish, Swahili, Swedish, Tagalog, Turkish, Vietnamese, Welsh
Malayalam	Malayalam
Mongolian	Mongolian
Oriya	Oriya
Sinhalese, Sinhala	Pali, Sinhalese
Syriac	Aramaic, Syriac
Tamil	Tamil
Telugu	Telugu
Thaana	Dhiveli, Divehi, Maldivian

Table D–3 Languages Supported by the World Lexer (Non-space-separated)

Language Group	Languages Include
Chinese	Cantonese, Mandarin, Pinyin phonograms
Japanese	Japanese (Hiragana, Kanji, Katakana)
Khmer	Cambodian, Khmer
Lao	Lao
Myanmar	Burmese
Thai	Thai
Tibetan	Dzongkha, Tibetan

Table D–4 shows languages not supported by the World Lexer.

Table D–4 Languages Not Supported by the World Lexer

Language Group	Languages Include
Buhid	Buhid
Canadian Syllabics	Blackfoot, Carrier, Cree, Dakhelh, Inuit, Inuktitut, Naskapi, Nunavik, Nunavut, Ojibwe, Sayisi, Slavey
Cherokee	Cherokee
Cypriot	Cypriot
Limbu	Limbu
Ogham	Ogham
Runic	Runic
Tai Le (Tai Lu, Lue, Dai Le)	Tai Le
Ugaritic	Ugaritic
Yi	Yi
Yi Jang Hexagram	Yi Jang

Querying

Oracle Text supports the use of different query operators. Some operators can be set to behave in accordance with your language. This section summarizes the multilingual query features for these operators.

ABOUT Operator

Use the `ABOUT` operator to query on concepts. The system looks up concept information in the theme component of the index.

This feature is supported for English and French with `CONTEXT` indexes only.

Fuzzy Operator

This operator enables you to search for words that have similar spelling to specified word. Oracle Text supports `fuzzy` for English, French, German, Italian, Dutch, Spanish, Portuguese, Japanese, Optical Character recognition (OCR), and automatic language detection.

Stem Operator

This operator enables you to search for words that have the same root as the specified term. For example, a stem of *$sing* expands into a query on the words *sang*, *sung*, *sing*. The Oracle Text stemmer supports the following languages: English, French, Spanish, Italian, German, Japanese and Dutch.

Supplied Stop Lists

A stoplist is a list of words that do not get indexed. These are usually common words in a language such as *this*, *that*, and *can* in English.

Oracle Text provides a default stoplist for English, Chinese (traditional and simplified), Danish, Dutch, Finnish, French, German, Italian, Portuguese, Spanish, and Swedish. Appendix E, "Oracle Text Supplied Stoplists", lists the stoplists for various languges.

Knowledge Base

An Oracle Text knowledge base is a hierarchical tree of concepts used for theme indexing, `ABOUT` queries, and deriving themes for document services.

Oracle Text supplies knowledge bases in English and French only.

Knowledge Base Extension

Extend theme functionality to languages other than English or French by loading your own knowledge base for any single byte white space delimited language, including Spanish.

Multilingual Features Matrix

The following table summarizes the multilingual features for the supported languages.

Table D–5 Multilingual Features for Supported Languages

LANGUAGE	ALTERNATE SPELLING	FUZZY MATCHING	LANGUAGE SPECIFIC LEXER	DEFAULT STOP LIST	STEMMING
ENGLISH	N/A	Yes	Yes	Yes	Yes
GERMAN	Yes	Yes	Yes	Yes	Yes
JAPANESE	N/A	Yes	Yes	No	Yes
FRENCH	N/A	Yes	Yes	Yes	Yes
SPANISH	N/A	Yes	Yes	Yes	Yes
ITALIAN	N/A	Yes	Yes	Yes	Yes
DUTCH	N/A	Yes	Yes	Yes	Yes
PORTUGUESE	N/A	Yes	Yes	Yes	Yes
KOREAN	N/A	No	Yes	No	Yes
SIMPLIFIED CHINESE	N/A	No	Yes	Yes	Yes
TRADITIONAL CHINESE	N/A	No	Yes	Yes	Yes
DANISH	Yes	No	Yes	No	Yes
SWEDISH	Yes	No	Yes	Yes	Yes
FINNISH	N/A	No	Yes	No	Yes
ARABIC	N/A	No	Yes	No	Yes
GREEK	N/A	No	Yes	No	Yes
BOKMAL	N/A	No	Yes	No	Yes
POLISH	N/A	No	Yes	No	Yes
RUSSIAN	N/A	No	Yes	No	Yes
SLOVENIAN	N/A	No	Yes	No	Yes
THAI	N/A	No	Yes	No	Yes
CATALAN	N/A	No	Yes	No	Yes
CROATIAN	N/A	No	Yes	No	Yes
HEBREW	N/A	No	Yes	No	Yes

Table D–5 (Cont.) Multilingual Features for Supported Languages

LANGUAGE	ALTERNATE SPELLING	FUZZY MATCHING	LANGUAGE SPECIFIC LEXER	DEFAULT STOP LIST	STEMMING
NYNORSK	N/A	No	Yes	No	Yes
SERBIAN	N/A	No	Yes	No	Yes
TURKISH	N/A	No	Yes	No	Yes
CZECH	N/A	No	Yes	No	Yes
HUNGARIAN	N/A	No	Yes	No	Yes
PERSIAN	N/A	No	Yes	No	Yes
SLOVAK	N/A	No	Yes	No	Yes

E

Oracle Text Supplied Stoplists

This appendix describes the default stoplists for all the different languages supported and lists the stopwords in each. The following stoplists are described:

English Default Stoplist

The following English words are defined as stop words:

Stopword	Stopword	Stopword	Stopword	Stopword	Stopword
a	did	in	only	then	where
all	do	into	onto	there	whether
almost	does	is	or	therefore	which
also	either	it	our	these	while
although	for	its	ours	they	who
an	from	just	s	this	whose
and	had	ll	shall	those	why
any	has	me	she	though	will
are	have	might	should	through	with
as	having	Mr	since	thus	would

Stopword	Stopword	Stopword	Stopword	Stopword	Stopword
at	he	Mrs	so	to	yet
be	her	Ms	some	too	you
because	here	my	still	until	your
been	hers	no	such	ve	yours
both	him	non	t	very	
but	his	nor	than	was	
by	how	not	that	we	
can	however	of	the	were	
could	i	on	their	what	
d	if	one	them	when	

Chinese Stoplist (Traditional)

The following traditional Chinese words are defined in the default stoplist for this language.

目前	由於	因此	他們	可能	沒有	希望
有關	不過	可以	如果	對於	因為	是否
但是	相當	其中	其他	雖然	我們	包括
必須	以上	之後	所以	以及	許多	最近
至於	一般	不是	不能	而且	引起	如何
除了	不少	最後	就是	分別	加強	甚至
繼續	另外	共同	只有	了解	根據	已經
過去	所有	不會	以來	任何	一直	不同
立即	左右	經過	尤其	使得	相關	當時
進入	並不	據了解	現在	只是	需要	原因
只要	否則	並未	什麼	如此	不要	

Chinese Stoplist (Simplified)

The following simplified Chinese words are defined in the default stoplist for this language.

必将	必须	并非	由于	一同	一再	一得
超过	成为	除了	处在	此项	从而	存在着
达到	大量	带来	带着	但是	当时	得到
都是	对于	这个	而且	而言	方面	各方面
各种	共同	还将	还有	很少	很有	还是
回到	获得了	或者	基本上	基于	即可	较大
尽管	就是	具有	可能	可以	来自	两个
之一	没有	目前	哪里	那里	却是	如果
如何	什么	实在	所需	所有	它的	他们
为了	我们	下去	现在	相当	新的	许多
也是	以及	已经	以上	因此	因为	

Danish (dk) Default Stoplist

The following Danish words are defined in the default stoplist for this language:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
af	en	god	hvordan	med	og	udenfor
aldrig	et	han	I	meget	oppe	under
alle	endnu	her	De	mellem	på	ved
altid	få	hos	i	mere	rask	vi
bagved	lidt	hovfor	imod	mindre	hurtig	
de	fjernt	hun	ja	når	sammen	
der	for	hvad	jeg	hvonår	temmelig	
du	foran	hvem	langsom	nede	nok	
efter	fra	hvor	mange	nej	til	
eller	gennem	hvorhen	måske	nu	uden	

Dutch (nl) Default Stoplist

The following Dutch words are defined in the default stoplist for this language:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
aan	betreffende	eer	had	juist	na	overeind	van	weer
aangaande	bij	eerdat	hadden	jullie	naar	overigens	vandaan	weg
aangezien	binnen	eerder	hare	kan	nadat	pas	vanuit	wegens
achter	binnenin	eerlang	heb	klaar	net	precies	vanwege	wel
achterna	boven	eerst	hebben	kon	niet	reeds	veeleer	weldra
afgelopen	bovenal	elk	hebt	konden	noch	rond	verder	welk
al	bovendien	elke	heeft	krachtens	nog	rondom	vervolgens	welke
aldaar	bovengenoemd	en	hem	kunnen	nogal	sedert	vol	wie
aldus	bovenstaand	enig	hen	kunt	nu	sinds	volgens	wiens
alhoewel	bovenvermeld	enigszins	het	later	of	sindsdien	voor	wier
alias	buiten	enkel	hierbeneden	liever	ofschoon	slechts	vooraf	wij
alle	daar	er	hierboven	maar	om	sommige	vooral	wijzelf

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
allebei	daarheen	erdoor	hij	mag	omdat	spoedig	vooralsnog	zal
alleen	daarin	even	hoe	meer	omhoog	steeds	voorbij	ze
alsnog	daarna	eveneens	hoewel	met	omlaag	tamelijk	voordat	zelfs
altijd	daarnet	evenwel	hun	mezelf	omstreeks	tenzij	voordezen	zichzelf
altoos	daarom	gauw	hunne	mij	omtrent	terwijl	voordien	zij
ander	daarop	gedurende	ik	mijn	omver	thans	voorheen	zijn
andere	daarvanlangs	geen	ikzelf	mijnent	onder	tijdens	voorop	zijne
anders	dan	gehad	in	mijner	ondertussen	toch	vooruit	zo
anderszins	dat	gekund	inmiddels	mijzelf	ongeveer	toen	vrij	zodra
behalve	de	geleden	inzake	misschien	ons	toenmaals	vroeg	zonder
behoudens	die	gelijk	is	mocht	onszelf	toenmalig	waar	zou
beide	dikwijls	gemoeten	jezelf	mochten	onze	tot	waarom	zouden
beiden	dit	gemogen	jij	moest	ook	totdat	wanneer	zowat
ben	door	geweest	jijzelf	moesten	op	tussen	want	zulke
beneden	doorgaand	gewoon	jou	moet	opnieuw	uit	waren	zullen
bent	dus	gewoonweg	jouw	moeten	opzij	uitgezonderd	was	zult
bepaald	echter	haar	jouwe	mogen	over	vaak	wat	

Finnish (sf) Default Stoplist

The following Finnish words are defined in the default stoplist for this language:

Stopword	Stopword	Stopword	Stopword	Stopword
ään	jälkeen	kumpi	nopeasti	suoraan
ah	jo	kumpikaan	nuo	ta
ai	joka	kumpikin	nyt	tä
aina	jokainen	kun	oi	tähden
alla	joku	kunhan	olemme	tahi
alle	jollei	kunnes	olen	tai
alta	jolleivat	kuten	olet	taikka
ansiosta	jollemme	kyllä	olette	takana
edessä	jollen	kylliksi	oli	takia
een	jollet	lähellä	olimme	tämä
ehkä	jollette	läpi	olin	tarpeeksi
ei	jos	liian	olit	tässä
eli	joskin	lla	olitte	te
elikkä	jotta	llä	olivat	tokko
ellei	kaikki	lle	ollut	tta
elleivät	kanssa	lta	on	ttä
ellemme	kaukana	ltä	oon	tuo
ellen	ken	luona	ovat	ulkopuolella

Stopword	Stopword	Stopword	Stopword	Stopword
ellet	keneksi	me	paitsi	useammin
ellette	kenelle	mikä	paljon	useimmin
enemmän	kenkään	mikään	paremmin	usein
eniten	kenties	mikäli	parhaiten	vaan
ennen	keskellä	mikin	pian	vähän
eräs	kesken	miksi	se	vähemmän
että	ketkä	milloin	seen	vähiten
hän	kohti	milloinkaan	sekä	vaikka
harva	koska	minä	sen	vailla
he	koskaan	missä	siellä	varten
hei	ksi	miten	sieltä	vastaan
hitaasti	kuin	molemmat	siin	vielä
hyi	kuinka	mutta	sillä	voi
hyvin	kuka	na	sinä	ympäri
iin	kukaan	nä	sinne	
ilman	kukin	näin	ssa	
itse	kumpainen	nämä	ssä	
ja	kumpainenkaan	ne	sta	
jahka	kumpainenkin	niin	stä	

French (f) Default Stoplist

The following French words are defined in the default stoplist for this language:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	beaucoup	comment	encore	lequel	moyennant	près	ses	toujours
afin	ça	concernant	entre	les	ne	puis	sien	tous
ailleurs	ce	dans	et	lesquelles	ni	puisque	sienne	toute
ainsi	ceci	de	étaient	lesquels	non	quand	siennes	toutes
alors	cela	dedans	était	leur	nos	quant	siens	très
après	celle	dehors	étant	leurs	notamment	que	soi	trop
attendant	celles	déjà	etc	lors	notre	quel	soi-même	tu
au	celui	delà	eux	lorsque	notres	quelle	soit	un
aucun	cependant	depuis	furent	lui	nôtre	quelqu'un	sont	une
aucune	certain	des	grâce	ma	nôtres	quelqu'une	suis	vos
au-dessous	certaine	desquelles	hormis	mais	nous	quelque	sur	votre
au-dessus	certaines	desquels	hors	malgré	nulle	quelques-unes	ta	vôtre
auprès	certains	dessus	ici	me	nulles	quelques-uns	tandis	vôtres
auquel	ces	dès	il	même	on	quels	tant	vous
aussi	cet	donc	ils	mêmes	ou	qui	te	vu

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
aussitôt	cette	donné	jadis	mes	où	quiconque	telle	y
autant	ceux	dont	je	mien	par	quoi	telles	
autour	chacun	du	jusqu	mienne	parce	quoique	tes	
aux	chacune	duquel	jusque	miennes	parmi	sa	tienne	
auxquelles	chaque	durant	la	miens	plus	sans	tiennes	
auxquels	chez	elle	laquelle	moins	plusieurs	sauf	tiens	
avec	combien	elles	là	moment	pour	se	toi	
à	comme	en	le	mon	pourquoi	selon	ton	

German (d) Default Stoplist

The following German words are defined in the default stoplist for this language:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
ab	dann	des	es	ihnen	keinem	obgleich	sondern	welchem
aber	daran	desselben	etwa	ihr	keinen	oder	sonst	welchen
allein	darauf	dessen	etwas	ihre	keiner	ohne	soviel	welcher
als	daraus	dich	euch	Ihre	keines	paar	soweit	welches
also	darin	die	euer	ihrem	man	sehr	über	wem
am	darüber	dies	eure	Ihrem	mehr	sei	um	wen
an	darum	diese	eurem	ihren	mein	sein	und	wenn
auch	darunter	dieselbe	euren	Ihren	meine	seine	uns	wer
auf	das	dieselben	eurer	Ihrer	meinem	seinem	unser	weshalb
aus	dasselbe	diesem	eures	ihrer	meinen	seinen	unsre	wessen
außer	daß	diesen	für	ihres	meiner	seiner	unsrem	wie
bald	davon	dieser	fürs	Ihres	meines	seines	unsren	wir
bei	davor	dieses	ganz	im	mich	seit	unsrer	wo
beim	dazu	dir	gar	in	mir	seitdem	unsres	womit
bin	dazwischen	doch	gegen	ist	mit	selbst	vom	zu
bis	dein	dort	genau	ja	nach	sich	von	zum
bißchen	deine	du	gewesen	je	nachdem	Sie	vor	zur
bist	deinem	ebenso	her	jedesmal	nämlich	sie	während	zwar
da	deinen	ehe	herein	jedoch	neben	sind	war	zwischen
dabei	deiner	ein	herum	jene	nein	so	wäre	
dadurch	deines	eine	hin	jenem	nicht	sogar	wären	
dafür	dem	einem	hinter	jenen	nichts	solch	warum	
dagegen	demselben	einen	hintern	jener	noch	solche	was	
dahinter	den	einer	ich	jenes	nun	solchem	wegen	
damit	denn	eines	ihm	kaum	nur	solchen	weil	
danach	der	entlang	ihn	kein	ob	solcher	weit	
daneben	derselben	er	Ihnen	keine	ober	solches	welche	

Italian (i) Default Stoplist

The following Italian words are defined in the default stoplist for this language:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	da	durante	lo	o	seppure	un
affinchè	dachè	e	loro	onde	si	una
agl'	dagl'	egli	ma	oppure	siccome	uno
agli	dagli	eppure	mentre	ossia	sopra	voi
ai	dai	essere	mio	ovvero	sotto	vostro
al	dal	essi	ne	per	su	
all'	dall'	finché	neanche	perchè	subito	
alla	dalla	fino	negl'	perciò	sugl'	
alle	dalle	fra	negli	però	sugli	
allo	dallo	giacchè	nei	poichè	sui	
anzichè	degl'	gl'	nel	prima	sul	
avere	degli	gli	nell'	purchè	sull'	
bensì	dei	grazie	nella	quand'anche	sulla	
che	del	I	nelle	quando	sulle	
chi	dell'	il	nello	quantunque	sullo	
cioè	delle	in	nemmeno	quasi	suo	
come	dello	inoltre	neppure	quindi	talchè	
comunque	di	io	noi	se	tu	
con	dopo	l'	nonchè	sebbene	tuo	
contro	dove	la	nondimeno	sennonchè	tuttavia	
cosa	dunque	le	nostro	senza	tutti	

Portuguese (pt) Default Stoplist

The following Portuguese words are defined in the default stoplist for this language:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	bem	e	longe	para	se	você
abaixo	com	ela	mais	por	sem	vocês
adiante	como	elas	menos	porque	sempre	
agora	contra	êle	muito	pouco	sim	
ali	debaixo	eles	não	próximo	sob	
antes	demais	em	ninguem	qual	sobre	
aqui	depois	entre	nós	quando	talvez	
até	depressa	eu	nunca	quanto	todas	
atras	devagar	fora	onde	que	todos	
bastante	direito	junto	ou	quem	vagarosamente	

Spanish (e) Default Stoplist

The following Spanish words are defined in the default stoplist for this language:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	aquí	cuantos	esta	misma	nosotras	querer	tales	usted
acá	cada	cuán	estar	mismas	nosotros	qué	tan	ustedes
ahí	cierta	cuánto	estas	mismo	nuestra	quien	tanta	varias
ajena	ciertas	cuántos	este	mismos	nuestras	quienes	tantas	varios
ajenas	cierto	de	estos	mucha	nuestro	quienesquiera	tanto	vosotras
ajeno	ciertos	dejar	hacer	muchas	nuestros	quienquiera	tantos	vosotros
ajenos	como	del	hasta	muchísima	nunca	quién	te	vuestra
al	cómo	demasiada	jamás	muchísimas	os	ser	tener	vuestras
algo	con	demasiadas	junto	muchísimo	otra	si	ti	vuestro
alguna	conmigo	demasiado	juntos	muchísimos	otras	siempre	toda	vuestros
algunas	consigo	demasiados	la	mucho	otro	sí	todas	y
alguno	contigo	demás	las	muchos	otros	sín	todo	yo
algunos	cualquier	el	lo	muy	para	Sr	todos	
algún	cualquiera	ella	los	nada	parecer	Sra	tomar	
allá	cualquieras	ellas	mas	ni	poca	Sres	tuya	
allí	cuan	ellos	más	ninguna	pocas	Sta	tuyo	
aquel	cuanta	él	me	ningunas	poco	suya	tú	
aquella	cuantas	esa	menos	ninguno	pocos	suyas	un	
aquellas	cuánta	esas	mía	ningunos	por	suyo	una	
aquello	cuántas	ese	mientras	no	porque	suyos	unas	
aquellos	cuanto	esos	mío	nos	que	tal	unos	

Swedish (s) Default Stoplist

The following Swedish words are defined in the default stoplist for this language:

Stopword	Stopword	Stopword	Stopword	Stopword
ab	du	hette	minst	skall
aldrig	efter	hon	minsta	skulle
all	efteråt	honom	mot	som
alla	eftersom	hos	mycket	ta
allt	ej	hur	någon	till
alltid	eller	i	någonting	tillräcklig
allting	emot	i fall	något	tillräckliga
än	en	ifall	några	tillräckligt
andra	endast	in	när	tillsammans
andre	er	inga	nära	tog
annan	era	ingen	ned	trots att
annat	ert	ingenting	nej	under
ännu	ett	inget	ner	underst
är	få	innan	nere	undre

Stopword	Stopword	Stopword	Stopword	Stopword
åter	fall	inte	ni	upp
att	färre	ja	nu	uppe
av	fastän	jag	och	ut
avse	flest	kan	också	utan
avsedd	flesta	kort	om	ute
avsedda	för	korta	oss	utom
avser	först	kunde	över	vad
avses	första	kunna	överst	väl
bakom	förste	lång	översta	var
bara	fort	långa	övre	vara
bäst	framför	långsam	på	varför
bättre	från	långsamma	så	vart
bra	genom	långsamt	sådan	vem
bredvid	god	långt	sådana	vems
då	goda	lite	sådant	vet
dålig	gott	liten	säga	veta
där	ha	litet	säger	vi
därför	hade	man	sägs	vid
de	haft	med	sämre	vilken
dem	han	medan	sämst	vill
den	hans	mellan	sån	ville
denna	här	men	sånt	visste
deras	hellre	mer	såsom	vore
dess	henne	mera	sin	
dessa	hennes	mest	sist	
det	heta	mesta	sista	
detta	heter	mindre	ska	

F

The Oracle Text Scoring Algorithm

This appendix describes how Oracle Text calculates scoring for word queries, which is different from the way it calculates scores for `ABOUT` queries in English. Scoring is obtained using the `SCORE` operator.

This appendix contains these topics:

- Scoring Algorithm for Word Queries
- Word Scoring Example
- DML and Scoring Algorithm

> **See Also:** "DEFINESCORE" on page 3-13 and "DEFINEMERGE" on page 3-12 for information about user-defined scoring

Scoring Algorithm for Word Queries

To calculate a relevance score for a returned document in a word query, Oracle Text uses an inverse frequency algorithm based on Salton's formula.

Inverse frequency scoring assumes that frequently occurring terms in a document set are noise terms, and so these terms are scored lower. For a document to score high, the query term must occur frequently in the document but infrequently in the document set as a whole.

The following table illustrates Oracle Text's inverse frequency scoring. The first column shows the number of documents in the document set, and the second column shows the number of terms in the document necessary to score 100.

This table assumes that only one document in the set contains the query term.

Number of Documents in Document Set	Occurrences of Term in Document Needed to Score 100
1	34
5	20
10	17
50	13
100	12
500	10
1,000	9
10,000	7

Number of Documents in Document Set	Occurrences of Term in Document Needed to Score 100
100,000	6
1,000,000	5

Note that the score varies, depending on the set size. For example, if only one document in the set contains the query term, and there are five documents in the set, then the term must occur 20 times in the document to score 100. If 1,000,000 documents are in the set, then the term can occur only 5 times in the document to score 100.

Word Scoring Example

You have 5000 documents dealing with chemistry in which the term *chemical* occurs at least once in every document. The term *chemical* thus occurs frequently in the document set.

You have a document that contains 5 occurrences of *chemical* and 5 occurrences of the term *hydrogen*. No other document contains the term *hydrogen*. The term hydrogen thus occurs infrequently in the document set.

Because *chemical* occurs so frequently in the document set, its score for the document is lower with respect to *hydrogen*, which is infrequent is the document set as a whole. The score for *hydrogen* is therefore higher than that of *chemical*. This is so even though both terms occur 5 times in the document.

Note: Even if the relatively infrequent term *hydrogen* occurred 4 times in the document, and *chemical* occurred 5 times in the document, the score for *hydrogen* might still be higher, because *chemical* occurs so frequently in the document set (at least 5000 times).

Inverse frequency scoring also means that adding documents that contain *hydrogen* lowers the score for that term in the document, and adding more documents that do not contain *hydrogen* raises the score.

DML and Scoring Algorithm

Because the scoring algorithm is based on the number of documents in the document set, inserting, updating or deleting documents in the document set is likely to change the score for any given term before and after DML.

If DML is heavy, you must optimize the index. Perfect relevance ranking is obtained by running a query right after optimizing the index.

If DML is light, Oracle Database still gives fairly accurate relevance ranking.

In either case, you must synchronize the index with CTX_DDL.SYNC_INDEX.

See Also: "SYNC_INDEX" on page 7-74

G

Oracle Text Views

This appendix lists all of the views provided by Oracle Text. The system provides the following views:

- CTX_CLASSES
- CTX_FILTER_BY_COLUMNS
- CTX_INDEXES
- CTX_INDEX_ERRORS
- CTX_INDEX_OBJECTS
- CTX_INDEX_PARTITIONS
- CTX_INDEX_SETS
- CTX_INDEX_SET_INDEXES
- CTX_INDEX_SUB_LEXERS
- CTX_INDEX_SUB_LEXER_VALUES
- CTX_INDEX_VALUES
- CTX_OBJECTS
- CTX_OBJECT_ATTRIBUTES
- CTX_OBJECT_ATTRIBUTE_LOV
- CTX_ORDER_BY_COLUMNS
- CTX_PARAMETERS
- CTX_PENDING
- CTX_PREFERENCES
- CTX_PREFERENCE_VALUES
- CTX_SECTIONS
- CTX_SECTION_GROUPS
- CTX_SQES
- CTX_STOPLISTS
- CTX_STOPWORDS
- CTX_SUB_LEXERS
- CTX_THESAURI

CTX_CLASSES

This view displays all the preference categories registered in the Text data dictionary. It can be queried by any user.

Column Name	Type	Description
CLA_NAME	VARCHAR2(30)	Class name
CLA_DESCRIPTION	VARCHAR2(80)	Class description

CTX_FILTER_BY_COLUMNS

This view displays all FILTER BY columns registered in the Text data dictionary. It can be queried by any user.

Column Name	Type	Description
FBC_INDEX_OWNER	VARCHAR2(30)	Index owner name
FBC_INDEX_NAME	VARCHAR2(30)	Index name
FBC_TABLE_OWNER	VARCHAR2(30)	Table owner name
FBC_TABLE_NAME	VARCHAR2(30)	Table name
FBC_COLUMN_NAME	VARCHAR2(256)	Column name
FBC_COLUMN_TYPE	VARCHAR2(30)	Column type
FBC_SECTION_TYPE	VARCHAR2(30)	Section type
FBC_SECTION_NAME	VARCHAR2(30)	Section name
FBC_SECTION_ID	NUMBER	Section ID

CTX_INDEXES

This view displays all indexes that are registered in the Text data dictionary for the current user. It can be queried by CTXSYS.

Column Name	Type	Description
IDX_CHARSET_COLUMN	VARCHAR2(256)	Name of the charset column in base table
IDX_DOCID_COUNT	NUMBER	Number of documents indexed
IDX_FORMAT_COLUMNS	VARCHAR2(256)	Name of the format column in base table
IDX_ID	NUMBER	Internal index ID
IDX_KEY_NAME	VARCHAR2(256)	Primary key column(s)
IDX_LANGUAGE_COLUMN	VARCHAR2(256)	Name of the language column in base table
IDX_NAME	VARCHAR2(30)	Name of index
IDX_OWNER	VARCHAR2(30)	Owner of index
IDX_STATUS	VARCHAR2(12)	Status
IDX_SYNC_INTERVAL	VARCHAR2(2000)	Interval string required by scheduler job. Only meaningful for AUTOMATIC sync. Always null for MANUAL and ON COMMIT sync.
IDX_SYNC_JOBNAME	VARCHAR2(50)	Scheduler job name for automatic sync. Only meaningful for AUTOMATIC sync and always null for other types of sync.
IDX_SYNC_MEMORY	VARCHAR2(100)	Sync memory size. Only meaningful for ON COMMIT and AUTOMATIC types of sync. For MANUAL sync, this is always null.
IDX_SYNC_PARA_DEGREE	NUMBER	Degree of parallelism for sync. Only meaningful for the AUTOMATIC type of sync; always null for MANUAL and ON COMMIT syncs.
IDX_SYNC_TYPE	VARCHAR2(20)	Type of synching: MANUAL, AUTOMATIC, or ON COMMIT
IDX_TABLE	VARCHAR2(30)	Table name
IDX_TABLE_OWNER	VARCHAR2(30)	Owner of table

Column Name	Type	Description
IDX_TEXT_NAME	VARCHAR2(30)	Text column name
IDX_TYPE	VARCHAR2(7)	Type of index: CONTEXT, CTXCAT, or CTXRULE

CTX_INDEX_ERRORS

This view displays the DML errors and is queryable by CTXSYS.

Column Name	Type	Description
ERR_INDEX_OWNER	VARCHAR2(30)	Index owner
ERR_INDEX_NAME	VARCHAR2(30)	Name of index
ERR_TIMESTAMP	DATE	Time of error
ERR_TEXTKEY	VARCHAR2(18)	ROWID of errored document or name of errored operation (for example, ALTER INDEX)
ERR_TEXT	VARCHAR2(4000)	Error text

CTX_INDEX_OBJECTS

This view displays the objects that are used for each class in the index. It can be queried by CTXSYS.

Column Name	Type	Description
IXO_INDEX_OWNER	VARCHAR2(30)	Index owner
IXO_INDEX_NAME	VARCHAR2(30)	Index name
IXO_CLASS	VARCHAR2(30)	Class name
IXO_OBJECT	VARCHAR2(30)	Object name

CTX_INDEX_PARTITIONS

This view displays all index partitions. It can be queried by CTXSYS.

Column Name	Type	Description
IXP_ID	NUMBER(38)	Index partition ID
IXP_INDEX_OWNER	VARCHAR2(30)	Index owner
IXP_INDEX_NAME	VARCHAR2(30)	Index name
IXP_INDEX_PARTITION_NAME	VARCHAR2(30)	Index partition name
IXP_SYNC_TYPE	VARCHAR2(20)	Type of synching: MANUAL, AUTOMATIC, or ON COMMIT
IXP_TABLE_OWNER	VARCHAR2(30)	Table owner
IXP_TABLE_NAME	VARCHAR2(30)	Table name

Column Name	Type	Description
IXP_TABLE_PARTITION_NAME	VARCHAR2(30)	Table partition name
IXP_DOCID_COUNT	NUMBER(38)	Number of documents associated with the partition
IXP_STATUS	VARCHAR2(12)	Partition status

CTX_INDEX_SETS

This view displays all index set names. It can be queried by any user.

Column Name	Type	Description
IXS_OWNER	VARCHAR2(30)	Index set owner
IXS_NAME	VARCHAR2(30)	Index set name

CTX_INDEX_SET_INDEXES

This view displays all the sub-indexes in an index set. It can be queried by any user.

Column Name	Type	Description
IXX_INDEX_SET_OWNER	VARCHAR2(30)	Index set owner
IXX_INDEX_SET_NAME	VARCHAR2(30)	Index set name
IXX_COLLIST	VARCHAR2(500)	Column list of the sub-index
IXX_STORAGE	VARCHAR2(500)	Storage clause of the sub-index

CTX_INDEX_SUB_LEXERS

This view shows the sub-lexers for each language for each index. It can be queried by CTXSYS.

Column Name	Type	Description
ISL_INDEX_OWNER	VARCHAR2(30)	Index owner
ISL_INDEX_NAME	VARCHAR2(30)	Index name
ISL_LANGUAGE	VARCHAR2(30)	Language of sub-lexer
ISL_ALT_VALUE	VARCHAR2(30)	Alternate value of language
ISL_OBJECT	VARCHAR2(30)	Name of lexer object used for this language

CTX_INDEX_SUB_LEXER_VALUES

Shows the sub-lexer attributes and their values. Accessible by CTXSYS.

Column Name	Type	Description
ISV_INDEX_OWNER	VARCHAR2(30)	Index owner
ISV_INDEX_NAME	VARCHAR2(30)	Index name
ISV_LANGUAGE	VARCHAR2(30)	Language of sub-lexer
ISV_OBJECT	VARCHAR2(30)	Name of lexer object used for this language
ISV_ATTRIBUTE	VARCHAR2(30)	Name of sub-lexer attribute
ISV_VALUE	VARCHAR2(500)	Value of attribute of sub-lexer

CTX_INDEX_VALUES

This view displays attribute values for each object used in indexes. This view is queryable by CTXSYS.

Column Name	Type	Description
IXV_INDEX_OWNER	VARCHAR2(30)	Index owner
IXV_INDEX_NAME	VARCHAR2(30)	Index name
IXV_CLASS	VARCHAR2(30)	Class name
IXV_OBJECT	VARCHAR2(30)	Object name
IXV_ATTRIBUTE	VARCHAR2(30)	Attribute name
IXV_VALUE	VARCHAR2(500)	Attribute value

CTX_OBJECTS

This view displays all of the Text objects registered in the Text data dictionary. This view can be queried by any user.

Column Name	Type	Description
OBJ_CLASS	VARCHAR2(30)	Object class (Datastore, Filter, Lexer, and so on)
OBJ_NAME	VARCHAR2(30)	Object name
OBJ_DESCRIPTION	VARCHAR2(80)	Object description

CTX_OBJECT_ATTRIBUTES

This view displays the attributes that can be assigned to preferences of each object. It can be queried by all users.

Column Name	Type	Description
OAT_CLASS	VARCHAR2(30)	Object class (Data Store, Filter, Lexer, and so on)
OAT_OBJECT	VARCHAR2(30)	Object name
OAT_ATTRIBUTE	VARCHAR2(64)	Attribute name
OAT_DESCRIPTION	VARCHAR2(80)	Description of attribute
OAT_REQUIRED	VARCHAR2(1)	Required attribute, either Y or N

Column Name	Type	Description
OAT_STATIC	VARCHAR2(1)	Not currently used
OAT_DATATYPE	VARCHAR2(64)	Attribute datatype. The value PROCEDURE indicates that the attribute of the object should be a stored procedure name.
OAT_DEFAULT	VARCHAR2(500)	Default value for attribute
OAT_MIN	NUMBER	Minimum value
OAT_MAX	NUMBER	Maximum value
OAT_MAX_LENGTH	NUMBER	Maximum length

CTX_OBJECT_ATTRIBUTE_LOV

This view displays the allowed values for certain object attributes provided by Oracle Text. It can be queried by all users.

Column Name	Type	Description
OAL_CLASS	NUMBER(38)	Class of object
OAL_OBJECT	VARCHAR2(30)	Object name
OAL_ATTRIBUTE	VARCHAR2(32)	Attribute name
OAl_LABEL	VARCHAR2(30)	Attribute value label
OAL_VALUE	VARCHAR2(64)	Attribute value
OAL_DESCRIPTION	VARCHAR2(80)	Attribute value description

CTX_ORDER_BY_COLUMNS

This view displays the ORDER BY columns registered in the Text data dictionary. It can be queried by any user.

Column Name	Type	Description
OBC_INDEX_OWNER	VARCHAR2(30)	Index owner
OBC_INDEX_NAME	VARCHAR2(30)	Index name
OBC_TABLE_OWNER	VARCHAR2(30)	Table owner
OBC_TABLE_NAME	VARCHAR2(30)	Table name
OBC_COLUMN_NAME	VARCHAR2(236)	Column name
OBC_COLUMN_POSITION	VARCHAR2(30)	Column position
OBC_COLUMN_TYPE	VARCHAR2(30)	Column type
OBC_SECTION_NAME	VARCHAR2(30)	Section name
OBC_SECTION_TYPE	VARCHAR2(30)	Section type
OBC_SECTION_ID	NUMBER	Section ID
OBC_SORT_ORDER	VARCHAR2(8)	Sort order

CTX_PARAMETERS

This view displays all system-defined parameters as defined by CTXSYS. It can be queried by any user.

Column Name	Type	Description
PAR_NAME	VARCHAR2(30)	Parameter name: max_index_memory ctx_doc_key_type default_index_memory default_datastore default_filter_binary default_filter_text default_filter_file default_section_html default_section_xml default_section_text default_lexer default_stoplist default_storage default_wordlist default_ctxcat_lexer default_ctxcat_index_set default_ctxcat_stoplist default_ctxcat_storage default_ctxcat_wordlist default_ctxrule_lexer default_ctxrule_stoplist default_ctxrule_storage default_ctxrule_wordlist log_directory file_access_role
PAR_VALUE	VARCHAR2(500)	Parameter value. For max_index_memory and default_index_memory, PAR_VALUE stores a string consisting of the memory amount. For the other parameter names, PAR_VALUE stores the names of the preferences used as defaults for index creation.

CTX_PENDING

This view displays a row for each of the user's entries in the DML Queue. It can be queried by CTXSYS.

Column Name	Type	Description
PND_INDEX_OWNER	VARCHAR2(30)	Index owner
PND_INDEX_NAME	VARCHAR2(30)	Name of index

Column Name	Type	Description
PND_PARTITION_NAME	VARCHAR2(30)	Name of partition for local partition indexes. NULL for normal indexes.
PND_ROWID	ROWID	ROWID to be indexed
PND_TIMESTAMP	DATE	Time of modification

CTX_PREFERENCES

This view displays preferences created by Oracle Text users, as well as all the system-defined preferences included with Oracle Text. The view contains one row for each preference. It can be queried by all users.

Column Name	Type	Description
PRE_OWNER	VARCHAR2(30)	Username of preference owner
PRE_NAME	VARCHAR2(30)	Preference name
PRE_CLASS	VARCHAR2(30)	Preference class
PRE_OBJECT	VARCHAR2(30)	Object used

CTX_PREFERENCE_VALUES

This view displays the values assigned to all the preferences in the Text data dictionary. The view contains one row for each value. It can be queried by all users.

Column Name	Type	Description
PRV_OWNER	VARCHAR2(30)	Username of preference owner
PRV_PREFERENCE	VARCHAR2(30)	Preference name
PRV_ATTRIBUTE	VARCHAR2(64)	Attribute name
PRV_VALUE	VARCHAR2(500)	Attribute value

CTX_SECTIONS

This view displays information about all the sections that have been created in the Text data dictionary. It can be queried by any user.

Column Name	Type	Description
SEC_OWNER	VARCHAR2(30)	Owner of the section group
SEC_SECTION_GROUP	VARCHAR2(30)	Name of the section group
SEC_TYPE	VARCHAR2(30)	Type of section, either ZONE, FIELD, SPECIAL, ATTR, STOP
SEC_ID	NUMBER	Section ID
SEC_NAME	VARCHAR2(30)	Name of section
SEC_TAG	VARCHAR2(64)	Section tag
SEC_VISIBLE	VARCHAR2(1)	Y or N visible indicator for field sections only

Column Name	Type	Description
SEC_DATATYPE	VARCHAR2(30)	Shows the datatype name (NUMBER, VARCHAR2, DATE or RAW) if the section is an SDATA section. Otherwise, it is NULL.

CTX_SECTION_GROUPS

This view displays information about all the section groups that have been created in the Text data dictionary. It can be queried by any user.

Column Name	Type	Description
SGP_OWNER	VARCHAR2(30)	Owner of section group
SGP_NAME	VARCHAR2(30)	Name of section group
SGP_TYPE	VARCHAR2(30)	Type of section group

CTX_SQES

This view displays the definitions for all SQEs that have been created by users. It can be queried by all users.

Column Name	Type	Description
SQE_OWNER	VARCHAR2(30)	Owner of SQE
SQE_NAME	VARCHAR2(30)	Name of SQE
SQE_QUERY	CLOB	Query Text

CTX_STOPLISTS

This view displays stoplists. Queryable by all users.

Column Name	Type	Description
SPL_OWNER	VARCHAR2(30)	Owner of stoplist
SPL_NAME	VARCHAR2(30)	Name of stoplist
SPL_COUNT	NUMBER	Number of stopwords
SPL_TYPE	VARCHAR2(30)	Type of stoplist, MULTI or BASIC

CTX_STOPWORDS

This view displays the stopwords in each stoplist. Queryable by all users.

Column Name	Type	Description
SPW_OWNER	VARCHAR2(30)	Stoplist owner
SPW_STOPLIST	VARCHAR2(30)	Stoplist name
SPW_TYPE	VARCHAR2(10)	Stop type, either STOP_WORD, STOP_CLASS, STOP_THEME
SPW_WORD	VARCHAR2(80)	Stopword
SPW_LANGUAGE	VARCHAR2(30)	Stopword language

CTX_SUB_LEXERS

This view contains information on multi-lexers and the sub-lexer preferences they contain. It can be queried by any user.

Column Name	Type	Description
SLX_OWNER	VARCHAR2(30)	Owner of the multi-lexer preference
SLX_NAME	VARCHAR2(30)	Name of the multi-lexer preference
SLX_LANGUAGE	VARCHAR2(30)	Language of the referenced lexer (full name, not abbreviation)
SLX_ALT_VALUE	VARCHAR2(30)	An alternate value for the language
SLX_SUB_OWNER	VARCHAR2(30)	Owner of the sub-lexer
SLX_SUB_NAME	VARCHAR2(30)	Name of the sub-lexer

CTX_THESAURI

This view displays information about all the thesauri that have been created in the Text data dictionary. It can be queried by any user.

Column Name	Type	Description
THS_OWNER	VARCHAR2(30)	Thesaurus owner
THS_NAME	VARCHAR2(30)	Thesaurus name

CTX_THES_PHRASES

This view displays phrase information for all thesauri in the Text data dictionary. It can be queried by any user.

Column Name	Type	Description
THP_THESAURUS	VARCHAR2(30)	Thesaurus name
THP_PHRASE	VARCHAR2(256)	Thesaurus phrase
THP_QUALIFIER	VARCHAR2(256)	Thesaurus qualifier
THP_SCOPE_NOTE	VARCHAR2(2000)	Thesaurus scope notes

CTX_TRACE_VALUES

This view contains one row for each active trace, and shows the current value of each trace.

Column Name	Type	Description
TRC_ID	BINARY_INTEGER	Trace ID
TRC_VALUE	NUMBER	Current trace value

Note: The error "ORA-00955: name is already used by an existing object" can safely be ignored if this error is raised in the post-install steps for patch releases. This may occur when this view is present in the database being patched.

CTX_USER_ FILTER_BY_COLUMNS

This view displays all FILTER BY columns registered in the Text data dictionary for the current user. It can be queried by any user.

Column Name	Type	Description
FBC_INDEX_NAME	VARCHAR2(30)	Index name
FBC_TABLE_OWNER	VARCHAR2(30)	Table owner name
FBC_TABLE_NAME	VARCHAR2(30)	Table name
FBC_COLUMN_NAME	VARCHAR2(256)	Column name
FBC_COLUMN_TYPE	VARCHAR2(30)	Column type
FBC_SECTION_TYPE	VARCHAR2(30)	Section type
FBC_SECTION_NAME	VARCHAR2(30)	Section name
FBC_SECTION_ID	NUMBER	Section ID

CTX_USER_INDEXES

This view displays all indexes that are registered in the Text data dictionary for the current user. It can be queried by all users.

Column Name	Type	Description
IDX_CHARSET_COLUMN	VARCHAR2(256)	Name of the charset column of base table
IDX_DOCID_COUNT	NUMBER	Number of documents indexed
IDX_FORMAT_COLUMN	VARCHAR2(256)	Name of the format column of base table
IDX_ID	NUMBER	Internal index ID
IDX_KEY_NAME	VARCHAR(256)	Primary key column(s)
IDX_LANGUAGE_COLUMN	VARCHAR2(256)	Name of the language column of base table
IDX_NAME	VARCHAR2(30)	Name of index
IDX_STATUS	VARCHAR2(12)	Status, either INDEXED or INDEXING
IDX_SYNC_INTERVAL	VARCHAR2(2000)	This is the interval string required by scheduler job. Only meaningful for AUTOMATIC sync. Always null for MANUAL and ON COMMIT sync.
IDX_SYNC_JOBNAME	VARCHAR2(50)	This is the scheduler job name for automatic sync. Only meaningful for AUTOMATIC sync and always null for other types of sync.

Column Name	Type	Description
IDX_SYNC_MEMORY	VARCHAR2(100)	The sync memory size. Only meaningful for ON COMMIT and AUTOMATIC types of sync. For MANUAL sync, this is always null.
IDX_SYNC_PARA_DEGREE	NUMBER	Degree of parallelism for sync. Only meaningful for the AUTOMATIC type of sync; always null for MANUAL and ON COMMIT syncs.
IDX_SYNC_TYPE	VARCHAR2(20)	Type of synching: AUTOMATIC, MANUAL or ON COMMIT
IDX_TABLE	VARCHAR2(30)	Table name
IDX_TABLE_OWNER	VARCHAR2(30)	Owner of table
IDX_TEXT_NAME	VARCHAR2(30)	Text column name
IDX_TYPE	VARCHAR2(30)	Type of index: CONTEXT, CTXCAT, or CTXRULE

CTX_USER_INDEX_ERRORS

This view displays the indexing errors for the current user and is queryable by all users.

Column Name	Type	Description
ERR_INDEX_NAME	VARCHAR2(30)	Name of index
ERR_TIMESTAMP	DATE	Time of error
ERR_TEXTKEY	VARCHAR2(18)	ROWID of errored document or name of errored operation (for example, ALTER INDEX)
ERR_TEXT	VARCHAR2(4000)	Error text

CTX_USER_INDEX_OBJECTS

This view displays the preferences that are attached to the indexes defined for the current user. It can be queried by all users.

Column Name	Type	Description
IXO_INDEX_NAME	VARCHAR2(30)	Name of index
IXO_CLASS	VARCHAR2(30)	Object name
IXO_OBJECT	VARCHAR2(80)	Object description

CTX_USER_INDEX_PARTITIONS

This view displays all index partitions for the current user. It is queryable by all users.

Column Name	Type	Description
IXP_DOCID_COUNT	NUMBER(38)	Number of documents associated with the index partition
IXP_ID	NUMBER(38)	Index partition ID
IXP_INDEX_NAME	VARCHAR2(30)	Index name
IXP_INDEX_PARTITION_NAME	VARCHAR2(30)	Index partition name
IDX_SYNC_INTERVAL	VARCHAR2(2000)	This is the interval string required by scheduler job. Only meaningful for AUTOMATIC sync. Always null for MANUAL and ON COMMIT sync.
IDX_SYNC_JOBNAME	VARCHAR2(50)	This is the scheduler job name for automatic sync. It is only meaningful for AUTOMATIC sync and always null for other types of sync.
IDX_SYNC_MEMORY	VARCHAR2(100)	The sync memory size. Only meaningful for ON COMMIT and AUTOMATIC types of sync. For MANUAL sync, this is always null.
IDX_SYNC_PARA_DEGREE	NUMBER	Degree of parallelism for sync. Only meaningful for the AUTOMATIC type of sync; always null for MANUAL and ON COMMIT syncs.
IDX_SYNC_TYPE	VARCHAR2(20)	Type of synching: AUTOMATIC, MANUAL or ON COMMIT
IXP_STATUS	VARCHAR2(12)	Partition status
IXP_TABLE_OWNER	VARCHAR2(30)	Table owner
IXP_TABLE_NAME	VARCHAR2(30)	Table name
IXP_TABLE_PARTITION_NAME	VARCHAR2(30)	Table partition name

CTX_USER_INDEX_SETS

This view displays all index set names that belong to the current user. It is queryable by all users.

Column Name	Type	Description
IXS_NAME	VARCHAR2(30)	Index set name

CTX_USER_INDEX_SET_INDEXES

This view displays all the indexes in an index set that belong to the current user. It is queryable by all users.

Column Name	Type	Description
IXX_INDEX_SET_NAME	VARCHAR2(30)	Index set name
IXX_COLLIST	VARCHAR2(500)	Column list of the index
IXX_STORAGE	VARCHAR2(500)	Storage clause of the index

CTX_USER_INDEX_SUB_LEXERS

This view shows the sub-lexers for each language for each index for the querying user. This view can be queried by all users.

Column Name	Type	Description
ISL_INDEX_NAME	VARCHAR2(30)	Index name
ISL_LANGUAGE	VARCHAR2(30)	Language of sub-lexer
ISL_ALT_VALUE	VARCHAR2(30)	Alternate value of language
ISL_OBJECT	VARCHAR2(30)	Name of lexer object used for this language

CTX_USER_INDEX_SUB_LEXER_VALS

Shows the sub-lexer attributes and their values for the querying user. This view can be queried by all users.

Column Name	Type	Description
ISV_INDEX_NAME	VARCHAR2(30)	Index name
ISV_LANGUAGE	VARCHAR2(30)	Language of sub-lexer
ISV_OBJECT	VARCHAR2(30)	Name of lexer object used for this language
ISV_ATTRIBUTE	VARCHAR2(30)	Name of sub-lexer attribute
ISV_VALUE	VARCHAR2(500)	Value of sub-lexer attribute

CTX_USER_INDEX_VALUES

This view displays attribute values for each object used in indexes for the current user. This view is queryable by all users.

Column Name	Type	Description
IXV_INDEX_NAME	VARCHAR2(30)	Index name
IXV_CLASS	VARCHAR2(30)	Class name
IXV_OBJECT	VARCHAR2(30)	Object name
IXV_ATTRIBUTE	VARCHAR2(30)	Attribute name
IXV_VALUE	VARCHAR2(500)	Attribute value

CTX_USER_ORDER_BY_COLUMNS

This view displays all ORDER BY columns registered in the Text data dictionary for the current user. It can be queried by any user.

Column Name	Type	Description
OBC_INDEX_NAME	VARCHAR2(30)	Index name
OBC_TABLE_OWNER	VARCHAR2(30)	Table owner
OBC_TABLE_NAME	VARCHAR2(30)	Table name
OBC_COLUMN_NAME	VARCHAR2(236)	Column name

Column Name	Type	Description
OBC_COLUMN_POSITION	VARCHAR2(30)	Column position
OBC_COLUMN_TYPE	VARCHAR2(30)	Column type
OBC_SECTION_NAME	VARCHAR2(30)	Section name
OBC_SECTION_TYPE	VARCHAR2(30)	Section type
OBC_SECTION_ID	NUMBER	Section ID
OBC_SORT_ORDER	VARCHAR2(8)	Sort order

CTX_USER_PENDING

This view displays a row for each of the user's entries in the DML Queue. It can be queried by all users.

Column Name	Type	Description
PND_INDEX_NAME	VARCHAR2(30)	Name of index
PND_PARTITION_NAME	VARCHAR2(30)	Name of partition for local partition indexes. NULL for normal indexes.
PND_ROWID	ROWID	Rowid to be indexed
PND_TIMESTAMP	DATE	Time of modification

CTX_USER_PREFERENCES

This view displays all preferences defined by the current user. It can be queried by all users.

Column Name	Type	Description
PRE_NAME	VARCHAR2(30)	Preference name
PRE_CLASS	VARCHAR2(30)	Preference class
PRE_OBJECT	VARCHAR2(30)	Object used

CTX_USER_PREFERENCE_VALUES

This view displays all the values for preferences defined by the current user. It can be queried by all users.

Column Name	Type	Description
PRV_PREFERENCE	VARCHAR2(30)	Preference name
PRV_ATTRIBUTE	VARCHAR2(64)	Attribute name
PRV_VALUE	VARCHAR2(500)	Attribute value

CTX_USER_SECTIONS

This view displays information about the sections that have been created in the Text data dictionary for the current user. It can be queried by all users.

Column Name	Type	Description
SEC_DATATYPE	VARCHAR2(30)	Shows the datatype name (NUMBER, VARCHAR2, DATE or RAW) if the section is an SDATA section. Otherwise, it is NULL.
SEC__SECTION_GROUP	VARCHAR2(30)	Name of the section group
SEC_TYPE	VARCHAR2(30)	Type of section, either ZONE, FIELD, SPECIAL, STOP, or ATTR
SEC_ID	NUMBER	Section ID
SEC_NAME	VARCHAR2(30)	Name of section
SEC_TAG	VARCHAR2(64)	Section tag
SEC_VISIBLE	VARCHAR2(1)	Y or N visible indicator for field sections

CTX_USER_SECTION_GROUPS

This view displays information about the section groups that have been created in the Text data dictionary for the current user. It can be queried by all users.

Column Name	Type	Description
SGP_NAME	VARCHAR2(30)	Name of section group
SGP_TYPE	VARCHAR2(30)	Type of section group

CTX_USER_SQES

This view displays the definitions of all persistent duration SQEs that have been created by the current user. In other words, it does not display session duration SQEs.

Column Name	Type	Description
SQE_OWNER	VARCHAR2(30)	Owner of SQE
SQE_NAME	VARCHAR2(30)	Name of SQE
SQE_QUERY	CLOB	Query text

CTX_USER_STOPLISTS

This view displays stoplists for current user. It is queryable by all users.

Column Name	Type	Description
SPL_NAME	VARCHAR2(30)	Name of stoplist
SPL_COUNT	NUMBER	Number of stopwords
SPL_TYPE	VARCHAR2(30)	Type of stoplist, MULTI or BASIC

CTX_USER_STOPWORDS

This view displays stopwords in each stoplist for current user. Queryable by all users.

Column Name	Type	Description
SPW_STOPLIST	VARCHAR2(30)	Stoplist name
SPW_TYPE	VARCHAR2(10)	Stop type, either STOP_WORD, STOP_CLASS, STOP_THEME
SPW_WORD	VARCHAR2(80)	Stopword
SPW_LANGUAGE	VARCHAR2(30)	Stopword language

CTX_USER_SUB_LEXERS

For the current user, this view contains information on multi-lexers and the sub-lexer preferences they contain. It can be queried by any user.

Column Name	Type	Description
SLX_NAME	VARCHAR2(30)	Name of the multi-lexer preference
SLX_LANGUAGE	VARCHAR2(30)	Language of the referenced lexer (full name, not abbreviation)
SLX_ALT_VALUE	VARCHAR2(30)	An alternate value for the language
SLX_SUB_OWNER	VARCHAR2(30)	Owner of the sub-lexer
SLX_SUB_NAME	VARCHAR2(30)	Name of the sub-lexer

CTX_USER_THESAURI

This view displays the information about all of the thesauri that have been created in the system by the current user. It can be viewed by all users.

Column Name	Type	Description
THS_NAME	VARCHAR2(30)	Thesaurus name

CTX_USER_THES_PHRASES

This view displays the phrase information of all thesauri owned by the current user. It can be queried by all users.

Column Name	Type	Description
THP_THESAURUS	VARCHAR2(30)	Thesaurus name
THP_PHRASE	VARCHAR2(256)	Thesaurus phrase
THP_QUALIFIER	VARCHAR2(256)	Phrase qualifier
THP_SCOPE_NOTE	VARCHAR2(2000)	Scope note of the phrase

CTX_VERSION

This view displays the CTXSYS data dictionary and code version number information.

Column Name	Type	Description
VER_DICT	CHAR(9)	The CTXSYS data dictionary version number

Column Name	Type	Description
VER_CODE	VARCHAR2(9)	The version number of the code linked in to the Oracle Database shadow process This column fetches the version number for linked-in code. Thus, use this column to detect and verify patch releases.

H

Stopword Transformations in Oracle Text

This appendix describes stopword transformations. The following topic is covered:

- Understanding Stopword Transformations

Understanding Stopword Transformations

When you use a stopword or stopword-only phrase as an operand for a query operator, Oracle Text rewrites the expression to eliminate the stopword or stopword-only phrase and then executes the query.

The following section describes the stopword rewrites or transformations for each operator. In all tables, the *Stopword Expression* column describes the query expression or component of a query expression, while the right-hand column describes the way Oracle Text rewrites the query.

The token *stopword* stands for a single stopword or a stopword-only phrase.

The token *non_stopword* stands for either a single non-stopword, a phrase of all non-stopwords, or a phrase of non-stopwords and stopwords.

The token *no_lex* stands for a single character or a string of characters that is neither a stopword nor a word that is indexed. For example, the + character by itself is an example of a *no_lex* token.

When the *Stopword Expression* column completely describes the query expression, a rewritten expression of *no_token* means that no hits are returned when you enter such a query.

When the *Stopword Expression* column describes a component of a query expression with more than one operator, a rewritten expression of *no_token* means that a *no_token* value is passed to the next step of the rewrite.

Transformations that contain a *no_token* as an operand in the *Stopword Expression* column describe intermediate transformations in which the *no_token* is a result of a previous transformation. These intermediate transformations apply when the original query expression has at least one stopword and more than one operator.

For example, consider the following compound query expression:

```
'(this NOT dog) AND cat'
```

Assuming that *this* is the only stopword in this expression, Oracle Text applies the following transformations in the following order:

stopword NOT non-stopword => no_token

no_token AND non_stopword => non_stopword

The resulting expression is:

```
'cat'
```

Word Transformations

Stopword Expression	Rewritten Expression
stopword	no_token
no_lex	no_token

The first transformation means that a stopword or stopword-only phrase by itself in a query expression results in no hits.

The second transformation says that a term that is not lexed, such as the + character, results in no hits.

AND Transformations

Stopword Expression	Rewritten Expression
non_stopword AND *stopword*	non_stopword
non_stopword AND *no_token*	non_stopword
stopword AND *non_stopword*	non_stopword
no_token AND *non_stopword*	non_stopword
stopword AND *stopword*	no_token
no_token AND *stopword*	no_token
stopword AND *no_token*	no_token
no_token AND *no_token*	no_token

OR Transformations

Stopword Expression	Rewritten Expression
non_stopword OR *stopword*	non_stopword
non_stopword OR *no_token*	non_stopword
stopword OR *non_stopword*	non_stopword
no_token OR *non_stopword*	non_stopword
stopword OR *stopword*	no_token
no_token OR *stopword*	no_token
stopword OR *no_token*	no_token
no_token OR *no_token*	no_token

ACCUMulate Transformations

Stopword Expression	Rewritten Expression
non_stopword ACCUM *stopword*	non_stopword
non_stopword ACCUM *no_token*	non_stopword
stopword ACCUM *non_stopword*	non_stopword
no_token ACCUM *non_stopword*	non_stopword
stopword ACCUM *stopword*	no_token
no_token ACCUM *stopword*	no_token
stopword ACCUM *no_token*	no_token
no_token ACCUM *no_token*	no_token

MINUS Transformations

Stopword Expression	Rewritten Expression
non_stopword MINUS *stopword*	non_stopword
non_stopword MINUS *no_token*	non_stopword
stopword MINUS *non_stopword*	no_token
no_token MINUS *non_stopword*	no_token
stopword MINUS *stopword*	no_token
no_token MINUS *stopword*	no_token
stopword MINUS *no_token*	no_token
no_token MINUS *no_token*	no_token

MNOT Transformations

Stopword Expression	Rewritten Expression
non_stopword MNOT *stopword*	non_stopword
non_stopword MNOT *no_token*	non_stopword
stopword MNOT *non_stopword*	no_token
no_token MNOT *non_stopword*	no_token
stopword MNOT *stopword*	no_token
no_token MNOT *stopword*	no_token
stopword MNOT *no_token*	no_token
no_token MNOT *no_token*	no_token

NOT Transformations

Stopword Expression	Rewritten Expression
non_stopword NOT *stopword*	non_stopword

Stopword Expression	Rewritten Expression
non_stopword NOT *no_token*	non_stopword
stopword NOT *non_stopword*	no_token
no_token NOT *non_stopword*	no_token
stopword NOT *stopword*	no_token
no_token NOT *stopword*	no_token
stopword NOT *no_token*	no_token
no_token NOT *no_token*	no_token

EQUIValence Transformations

Stopword Expression	Rewritten Expression
non_stopword EQUIV *stopword*	non_stopword
non_stopword EQUIV *no_token*	non_stopword
stopword EQUIV *non_stopword*	non_stopword
no_token EQUIV *non_stopword*	non_stopword
stopword EQUIV *stopword*	no_token
no_token EQUIV *stopword*	no_token
stopword EQUIV *no_token*	no_token
no_token EQUIV *no_token*	no_token

Note: When you use query explain plan, not all of the equivalence transformations are represented in the `EXPLAIN` table.

NEAR Transformations

Stopword Expression	Rewritten Expression
non_stopword NEAR *stopword*	non_stopword
non_stopword NEAR *no_token*	non_stopword
stopword NEAR *non_stopword*	non_stopword
no_token NEAR *non_stopword*	non_stopword
stopword NEAR *stopword*	no_token
no_token NEAR *stopword*	no_token
stopword NEAR *no_token*	no_token
no_token NEAR *no_token*	no_token

Weight Transformations

Stopword Expression	Rewritten Expression
stopword * n	no_token
no_token * n	no_token

Threshold Transformations

Stopword Expression	Rewritten Expression
stopword > n	no_token
no_token > n	no_token

WITHIN Transformations

Stopword Expression	Rewritten Expression
stopword WITHIN *section*	no_token
no_token WITHIN *section*	no_token

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z