Oracle® Text

Reference

Release 9.2

March 2002

Part No. A96518-01

ORACLE®

Oracle Text Reference, Release 9.2

Part No. A96518-01

Primary Author: Colin McGregor

Contributors: Omar Alonso, Shamim Alpha, Steve Buxton, Chung-Ho Chen, Jack Chen, Yun Cheng, Michele Cyran, Paul Dixon, Mohammad Faisal, Elena Huang, Garrett Kaminaga, Ji Sun Kang, Bryn Llewellyn, Wesley Lin, Yasuhiro Matsuda, Gerda Shank, and Steve Yang.

Contents

3 CONTAINS Query Operators

8 CTX_DOC Package

9 CTX_OUTPUT Package

10 CTX_QUERY Package

11 CTX_REPORT

12 CTX_THES Package

C Loading Examples

D Supplied Stoplists

E Alternate Spelling Conventions

F Scoring Algorithm

G Views

H Stopword Transformations

Send Us Your Comments

Oracle Text Reference, Release 9.2

Part No. A96518-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This manual provides reference information for Oracle Text. Use it as a reference for creating Oracle Text indexes, for issuing Oracle Text queries, for presenting documents, and for using the Oracle Text PL/SQL packages.

This preface contains these topics:

Audience

Oracle Text Reference is intended for an Oracle Text application developer or a system administrator responsible for maintaining the Oracle Text system.

To use this document, you need experience with the Oracle relational database management system, SQL, SQL*Plus, and PL/SQL. See the documentation provided with your hardware and software for additional information.

If you are unfamiliar with the Oracle RDBMS and related tools, read Chapter 1, “An Introduction to the Oracle Server”, in *Oracle9i Concepts*. The chapter is a comprehensive introduction to the concepts and terminology used throughout Oracle documentation.

Organization

This document contains:

Chapter 1, "SQL Statements and Operators"

This chapter describes the SQL statements and operators you can use with Oracle Text.

Chapter 2, "Indexing"

This chapter describes the indexing types you can use to create an Oracle Text index.

Chapter 3, "CONTAINS Query Operators"

This chapter describes the operators you can use in `CONTAINS` queries.

Chapter 4, "Special Characters in Queries"

This chapter describes the special characters you can use in `CONTAINS` queries.

Chapter 5, "CTX_ADM Package"

This chapter describes the procedures in the `CTX_ADM` PL/SQL package.

Chapter 7, "CTX_DDL Package"

This chapter describes the procedures in the `CTX_DDL` PL/SQL package. Use this package for maintaining your index.

Chapter 8, "CTX_DOC Package"

This chapter describes the procedures in the CTX_DOC PL/SQL package. Use this package for document services such as document presentation.

Chapter 9, "CTX_OUTPUT Package"

This chapter describes the procedures in the CTX_OUTPUT PL/SQL package. Use this package to manage your index error log files.

Chapter 10, "CTX_QUERY Package"

This chapter describes the procedures in the CTX_QUERY PL/SQL package. Use this package to manage queries such as to count hits and to generate query explain plan information.

Chapter 11, "CTX_REPORT"

This chapter describes the procedures in the CTX_REPORT PL/SQL package. Use this package to create various index reports.

Chapter 12, "CTX_THES Package"

This chapter describes the procedures in the CTX_THES PL/SQL package. Use this package to manage your thesaurus.

Chapter 13, "CTX_ULEXER Package"

This chapter describes the data types in the CTX_ULEXER PL/SQL package. Use this package with the user defined lexer.

Chapter 14, "Executables"

This chapter describes the supplied executables for Oracle Text including ctxload, the thesaurus loading program, and ctxkbtc, the knowledge base compiler.

Appendix A, "Result Tables"

This appendix describes the result tables for some of the procedures in CTX_DOC, CTX_QUERY, and CTX_THES packages.

Appendix B, "Supported Document Formats"

This appendix describes the supported document formats that can be filtered with the Inso filter for indexing.

Appendix C, "Loading Examples"
This appendix provides some basic examples for populating a text table.

Appendix D, "Supplied Stoplists"
This appendix describes the supplied stoplist for each supported language.

Appendix E, "Alternate Spelling Conventions"
This appendix describes the alternate spelling conventions used for German, Danish, and Swedish.

Appendix F, "Scoring Algorithm"
This appendix describes the scoring algorithm used for word queries.

Appendix G, "Views"
This appendix describes the Oracle Text views.

Appendix H, "Stopword Transformations"
This appendix describes stopword transformations.

Appendix I, "English Knowledge Base Category Hierarchy"
This appendix describes the supplied English Knowledge Base.

Related Documentation

For more information, see these Oracle resources:

For more information about *Oracle Text*, see:

- *Oracle Text Application Developer's Guide*

For more information about Oracle9*i*, see:

- *Oracle9i Concepts*
- *Oracle9i Administrator's Guide*
- *Oracle9i Database Utilities*
- *Oracle9i Performance Guide and Reference*

- *Oracle9i SQL Reference*
- *Oracle9i Reference*
- *Oracle9i Application Developer's Guide - Fundamentals*
- *Oracle9i Application Developer's Guide - XML*

For more information about PL/SQL, see:

- *PL/SQL User's Guide and Reference*

You can obtain Oracle Text technical information, collateral, code samples, training slides and other material at:

`http://otn.oracle.com/products/text/`

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/admin/account/membership.html`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/docs/index.htm`

To access the database documentation search engine directly, please visit

`http://tahiti.oracle.com`

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	The C datatypes such as **ub4**, **sword**, or **OCINumber** are valid. When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates query terms, book titles, emphasis, syntax clauses, or placeholders.	The following query searches for *oracle*. *Oracle9i Concepts* You can specify the *parallel_clause*. Run U*old_release*.SQL where *old_release* refers to the release you installed prior to upgrading.
UPPERCASE monospace (fixed-width font)	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, user names, and roles.	You can specify this clause only for a NUMBER column. You can back up the database using the BACKUP command. Query the TABLE_NAME column in the USER_TABLES data dictionary view. Specify the ROLLBACK_SEGMENTS parameter. Use the DBMS_STATS.GENERATE_STATS procedure.

Convention	Meaning	Example
lowercase monospace (fixed-width font)	Lowercase monospace typeface indicates executables and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, user names and roles, program units, and parameter values.	Enter `sqlplus` to open SQL*Plus. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (digits [ , precision ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`
...	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	`CREATE TABLE ... AS subquery;` `SELECT col1, col2, ... , coln FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	

Convention	Meaning	Example
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as it is shown.	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates variables for which you must supply particular values.	`CONNECT SYSTEM/system_password`

UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.	`SELECT last_name, employee_id FROM employees;` `sqlplus hr/hr`

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

What's New in Oracle Text?

This chapter describes new features of Oracle Text (formerly Oracle8*i* *inter*Media Text) and provides pointers to additional information. The following topics are covered:

- Release 9.2 New Features in Oracle Text
- Release 9.0.1 New Features in Oracle Text

Release 9.2 New Features in Oracle Text

The following features are new for this release:

- **Document Classification**

  The new CTX_CLS.TRAIN procedure enables you to generate rules for routing documents to different categories.

  See Also: TRAIN in Chapter 6, "CTX_CLS Package"

- **User Defined Lexer**

  The user-defined lexer enables you to create lexing solutions for indexing and querying languages not supported by Oracle Text such as Arabic.

  See Also: USER_LEXER in Chapter 2, "Indexing"

- **Query Templating**

  CONTAINS and CATSEARCH are no longer limited to their respective CONTEXT and CTXCAT grammars. Query templating enables you to use the CONTEXT grammar and associated operators in CATSEARCH queries and vice-versa.

  See Also: CATSEARCH in Chapter 1, "SQL Statements and Operators"

- **CREATE INDEX ONLINE Support**

  You can create a CONTEXT index while allowing inserts, updates, and deletes to your base table.

  See Also: CREATE INDEX in Chapter 1, "SQL Statements and Operators"

- **Parallel Indexing Enhancements**

  Parallel indexing is now supported for non-partitioned tables. You can use parallelism with `CREATE INDEX` and `ALTER INDEX` with parameters

`replace`, `resume`, and `sync`. You can also run `CTX_DDL.SYNC_INDEX` and `CTX_DDL.OPTIMIZE_INDEX` with a parallel degree.

See Also:

CREATE INDEX in Chapter 1, "SQL Statements and Operators"

SYNC_INDEX in Chapter 7, "CTX_DDL Package"

- **Stem Indexing**

  Stem indexing enables better performance for stem ($) queries by indexing the stem form in addition to the base form.

  See Also: BASIC_LEXER in Chapter 2, "Indexing"

- **Chinese Lexer**

  New CHINESE_LEXER enables you to index traditional and simplified Chinese text more efficiently.

  See Also: CHINESE_LEXER in Chapter 2, "Indexing"

- **URIType indexing**

  You can create CONTEXT indexes on URIType columns.

  See Also: CREATE INDEX in Chapter 1, "SQL Statements and Operators"

- **CTXXPATH**

  The CTXXPATH indextype enables you to speed up ExistsNode() queries on XMLType columns.

  See Also: Syntax for CTXXPATH Indextype in Chapter 1, "SQL Statements and Operators"

  Oracle9i Application Developer's Guide - XML

- **ORA:CONTAINS Support in ExistsNode()**

  You can call the CONTAINS function within an ExistsNode() statement without a Text index.

  See Also:

  Oracle9i Application Developer's Guide - XML

  CREATE_POLICY in Chapter 7, "CTX_DDL Package".

Release 9.0.1 New Features in Oracle Text

The following sections outline the new features in this release.

- **Document Classification**

  A document classification application is one that classifies an incoming stream of documents based on their content. These applications are also known as document routing or filtering applications. For example, an online news agency might need to classify its incoming stream of articles as they arrive into categories such as politics, crime, and sports.

  Oracle Text enables you to build such applications with the new `CTXRULE` index type. This index type indexes the rules (queries) that define classifications or routing criteria. When documents arrive, the new `MATCHES` operator can be used to categorize and route each document.

  Note: Oracle Text supports document classification for only plain text, XML, and HTML documents.

  See Also: CREATE INDEX and MATCHES statements in Chapter 1, "SQL Statements and Operators".

  Oracle Text Application Developer's Guide for more information about document classification.

- **Local Partitioned Index Support**

  You can create local partitioned indexes on partitioned text tables. To do so, use `CREATE INDEX` with the `LOCAL PARTITION` clause. You can also rebuild partitioned indexes with `ALTER INDEX`.

  See Also: CREATE INDEX and ALTER INDEX in Chapter 1, "SQL Statements and Operators".

- **IGNORE Format Column Value**

  The format column in your text table allows you to specify whether binary or text data is stored in the text column.

  A new format column value of `IGNORE` is provided. When you issue the `CREATE INDEX` statement and specify a format column, any row whose format column is set to `IGNORE` is ignored during indexing. This feature is useful for

indexing text columns that contain data incompatible with text indexing such as images or raw binary data.

See Also: CREATE INDEX in Chapter 1, "SQL Statements and Operators".

- **USER_DATASTORE Enhancement**

  When you specify your user procedure for the USER_DATSTORE, you can return permanent BLOB and CLOB locators for your IN/OUT parameter.

  See Also: USER_DATASTORE in Chapter 2, "Indexing".

- **New Korean Lexer**

  In this release, Oracle Text continues to support the indexing and querying of Korean text with a new Korean lexer, KOREAN_MORPH_LEXER. The KOREAN_MORPH_LEXER lexer offers the following benefits over the KOREAN_LEXER:

  - better morphological analysis of Korean text
  - faster indexing
  - smaller indexes
  - more accurate query searching

  See Also: KOREAN_MORPH_LEXER in Chapter 2, "Indexing".

- **New Japanese Lexer**

  In this release, Oracle Text continues to support the indexing and querying of Japanese text with a new Japanese lexer JAPANESE_LEXER. 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

  See Also: JAPANESE_LEXER in Chapter 2, "Indexing".

- **XMLType Indexing**

  Oracle Text supports the indexing of text columns of type `XMLType`.

  Note: `XMLType` indexing is supported only for the `CONTEXT` index type.

  See Also: *Oracle Text Application Developer's Guide* for more information about `XMLType` indexing.

- **All Language Stopwords**

  You can create a `MULTI_STOPLIST` type stoplist that contains words that are to be stopped in more than one language. This new stopword type is called ALL. For example, you can use an ALL stopword when you need to index international documents that contain English fragments.

  See Also: ADD_STOPWORD in Chapter 7, "CTX_DDL Package".

- **UTF-16 Auto-detection**

  Oracle Text supports UTF-16 conversion to the database character set with the charset and Inso filters. These filters can convert documents that are UTF-16 big-endian (AL16UTF16) or little-endian (AL16UTF16LE).

  Oracle Text also supports endian auto-detection when the character set column or charset filter is set to `UTF16AUTO`.

  See Also: CHARSET_FILTER in Chapter 2, "Indexing".

- **INSO_FILTER Timeout Attribute**

  The `INSO_FILTER` document filter has a new *timeout* attribute that allows you to specify the maximum time Oracle waits for a document to be filtered during indexing. You can use this mechanism to avoid hanging during the index operation.

  See Also: INSO_FILTER in Chapter 2, "Indexing".

- **XML Path Searching**

  XML documents can have parent-child tag structures such as the following:

  ```
  <A> <B> <C> dog </C> </B </A>
  ```

  In this example, tag C is a child of tag B which is a child of tag A.

  Oracle Text now enables you to do path searching with the new PATH_SECTION_GROUP. This section group allows you to specify direct parentage in queries, such as to find all documents that contain the term *dog* in element C which is a child of element B and so on.

  The new section group also allows you to do tag attribute value searching and attribute equality testing.

  The new operators associated with the this feature are

  - INPATH
  - HASPATH

  See Also: INPATH and HASPATH operators in Chapter 3, "CONTAINS Query Operators".

  Oracle Text Application Developer's Guide for more information about path section searching with XML documents.

- **CTX_DDL Updated Procedures**

  The following procedures in the CTX_DDL PL/SQL package have been updated:

  - CTX_DDL.SYNC_INDEX

    This procedure has two new parameters for specifying memory size and partition name.

  - CTX_DDL.SET_ATTRIBUTE

    See Also: Chapter 7, "CTX_DDL Package".

  This procedure accepts ON/OFF boolean attributes in addition to TRUE, T, FALSE,F, YES, Y, NO, and N.

- **CTX_DOC New Procedure**
  - CTX_DOC.IFILTER

    See Also: Chapter 8, "CTX_DOC Package".

  Use this procedure when you need your USER_DATASTORE procedure to filter binary data to text before concatenation.
- **CTX_OUTPUT New Procedures**

  The CTX_OUTPUT package has the following new procedures:
  - CTX_OUTPUT.ADD_EVENT
  - CTX_OUTPUT.REMOVE_EVENT

  Use the first procedure to augment the index log file with rowid information, which is useful for debugging an index operation.

  See Also: Chapter 9, "CTX_OUTPUT Package".

- **New and Updated Views**

  The following views have been updated for this release:
  - CTX_VERSION
  - CTX_PENDING
  - CTX_USER_PENDING

  The CTX_VERSION view has a new column VER_CODE which is the version number of the Oracle Text code linked in to the Oracle shadow process. Use this column to detect and verify patch releases.

  The following views are new. Use the first four for querying information about sub-lexers with multi-lexer preference:
  - CTX_INDEX_SUB_LEXERS
  - CTX_USER_INDEX_SUB_LEXERS
  - CTX_INDEX_SUB_LEXER_VALUES
  - CTX_USER_INDEX_SUB_LEXER_VALS
  - CTX_INDEX_PARTITIONS
  - CTX_USER_INDEX_PARTITIONS

- CTX_INDEX_SETS
- CTX_USER_INDEX_SETS
- CTX_INDEX_SET_INDEXES
- CTX_USER_INDEX_SET_INDEXES
- CTX_THES_PHRASES
- CTX_USER_THES_PHRASES

See Also: Appendix G, "Views".

1

SQL Statements and Operators

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

The following statements are described in this chapter:

- ALTER INDEX
- ALTER TABLE: Supported Partitioning Statements
- CATSEARCH
- CONTAINS
- CREATE INDEX
- CATSEARCH
- MATCHES
- SCORE

ALTER INDEX

> **Note:** This section describes the ALTER INDEX statement as it pertains to managing a Text domain index.
>
> For a complete description of the ALTER INDEX statement, see *Oracle9i SQL Reference*.

Purpose

Use ALTER INDEX to perform the following maintenance tasks for a CONTEXT, CTXCAT, or CTXRULE index:

All Indextypes

- Rename the index or index partition. See RENAME Syntax.
- Rebuild the index using different preferences. Some restrictions apply for the CTXCAT indextype. See REBUILD Syntax.
- Add stopwords to the index. See REBUILD Syntax.

CONTEXT and CTXRULE Indextypes

- Resume a failed index operation (creation / optimization). See REBUILD Syntax.
- Process DML in batch (synchronize). See REBUILD Syntax.
- Optimize the index, fully or by token. See REBUILD Syntax.
- Add sections and stop sections to the index. See REBUILD Syntax.

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. If you specify a name longer than 25 bytes, Oracle returns an error and the renamed index is no longer valid.

> **Note:** When `new_index_name` is more than 25 bytes and less than 30 bytes, Oracle 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.

REBUILD Syntax

The following syntax is used to rebuild the index, rebuild an index partition, resume a failed operation, perform batch DML, add stopwords to index, add sections and stop sections to index, or optimize the index:

```
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 `sync, optimize full/fast, resume` or `replace` in `paramstring`. These operations work only on the `partname` you specify. You cannot specify `resume` when you rebuild partitions or a partitioned index.

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 automatically creates the metadata for the new index partition. The new index partition has the same name as the new table partition. You can change the index partition name with `ALTER INDEX RENAME`. To populate the new index partition, you must rebuild it with `ALTER INDEX REBUILD`.

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

[ONLINE]
Optionally specify the `ONLINE` parameter for nonblocking operation, which allows the index to be queried during an `ALTER INDEX` synchronize or optimize operation.

You cannot use PARALLEL with ONLINE.

> **Note:** You can specify replace or resume when rebuilding and index ONLINE, but you cannot specify replace or resume when rebuilding and index partition ONLINE.

PARALLEL n
Optionally specify with n 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.

You cannot use PARALLEL with ONLINE.

PARAMETERS (paramstring)

Optionally specify paramstring. If you do not specify paramstring, Oracle 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]
    [index set index_set]

    |    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 preferences, your own or system-defined.

You can only replace 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 only lexer, wordlist, storage index set, and memory preferences.

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

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 optimizes only the index token index_token in token mode. Use this method of optimization to quickly optimize index information for specific words.

When you optimize in fast mode, Oracle 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 need to optimize time-limited batches with the maxtime parameter.

You use the maxtime parameter to specify in minutes the time Oracle is to spend on the optimization operation. Oracle 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 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 performs minimal optimization.

> **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 runtime memory to use with memsize. You 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 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.

When your stoplist is a multi-language 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: "Add Section Constraints" on page 1-9.

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.

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: "Add Section Constraints" on page 1-9.

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 section section_name applies only to documents indexed after this operation. Thus 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 indexes. It does not apply to CTXCAT indexes.

> **See Also:** "Add Section Constraints" in this section.

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 is always searchable.

The number of stop sections you can add is unlimited.

The index is *not* rebuilt by this statement.

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

Add Section Constraints

Before altering the index section information, Oracle 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 field sections.
- You cannot add stop sections to basic, HTML, XML, or news section groups.
- SENTENCE and PARAGRAPH are reserved section names.

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 then rebuilds the index with `ALTER 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));

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 it is already been used. In this case, Oracle 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') >0;
```

Optimizing the Index

Optimizing your index with `ALTER INDEX` will not be supported in future releases. To optimize your index, use `CTX_DDL.OPTIMIZE_INDEX`.

Synchronizing the Index

Synchronizing the index with `ALTER INDEX` will not be supported in future releases. 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>`, issue 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, issue 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>
```

You want to create a separate section for the title attribute and you want to name the new attribute section booktitle. To do so, issue the following statement:

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

Related Topics

CTX_DDL.SYNC_INDEX in Chapter 7, "CTX_DDL Package"

CTX_DDL.OPTIMIZE_INDEX in Chapter 7, "CTX_DDL Package"

CREATE INDEX

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 *Oracle9i SQL Reference*.

Purpose

You can use ALTER TABLE 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 you can issue.

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.

Rebuild Unusable Local Indexes

```
ALTER TABLE  [schema.]table MODIFY PARTITION partition REBUILD UNUSABLE 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 issue this command, this command does NOT rebuild the index partition. Do not depend on this command 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 at the beginning or in the middle of the table, use ALTER TABLE SPLIT PARTITION.

The newly added table partition is always empty, and the context domain index (if any) status for this partition is always VALID. After doing DML, if you want to synchronize or optimize this newly added index partition, you must look up the index partition name, and then issue the ALTER INDEX REBUILD PARTITION command. For this newly added partition, 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 (as its name), system will assign a name in the form of SYS_Pn.

By querying the USER_IND_PARTITIONS view and compare the HIGH_VALUE field, you can figure out 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]]
```

Applies only to a range partition. This command 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, the corresponding local domain index partition is marked UNUSABLE. Users can use ALTER TABLE MODIFY PARTITION to rebuild the partition index.

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

Split Partition Syntax

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

Applies only to range partition. This command 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. You can use ALTER TABLE MODIFY PARTITION to rebuild the partition indexes.

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

Exchange Partition Syntax

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

Converts a partition to a non-partitioned table and 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 this case, you can issue ALTER TABLE MODIFY PARTITION.

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

Field Sections

Field section queries might not work the same if the non-partitioned index and local index use different section id's for the same field section.

Storage

Storage is not changed. So if the index on the non-partitioned 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 the table may be created in a different location.

Restrictions

Both indexes must be equivalent. They must use the same objects, same settings for each object. Note: we only check that they 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. The user needs to manually rebuild both index and index partition using ALTER INDEX ... REBUILD and ALTER INDEX ... REBUILD.

Truncate Partition Syntax

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

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

CATSEARCH

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

The grammar of this operator is called CTXCAT. You can also use the CONTEXT grammar if your search criteria requires 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

You use the CATSEARCH operator with a CTXCAT index mainly to improve mixed query performance. You specify your text query condition with `text_query` and your structured condition with `structured_query`.

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

Limitation

This operator does not support functional invocation.

Syntax

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

[schema.]column
Specify 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

These operators have the following syntax:

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.

Operation	Syntax	Description of Operation
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. You set prefix indexing with the BASIC_WORDLIST preference.

Query Template Specification You specify a marked-up string that specifies a query based on the CONTEXT grammar. Use the following tags and attribute values which are case sensitive:

TAG	Description	Possible Values
<query> </query>	Signals that this query be interpreted as a query template.	
<textquery> </textquery>	Specify the query string.	
grammar=	Specify the grammar of the query.	CONTEXT CTXCAT
<score></score>	Specify the score preference	
datatype=	Specify the type of number returned as score.	INTEGER FLOAT

structured_query

Specify 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 `CTX_DDL.ADD_INDEX`.

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

- =

- <=
- >=
- >
- <
- IN
- BETWEEN

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

Examples

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');
```

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 CTXCAT PARAMETERS
('index set auction_iset');
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 Example

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

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

Related Topics

Syntax for CTXCAT Indextype in this chapter.

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.

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

The grammar for this operator is called CONTEXT. 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.

Syntax

```
CONTAINS(
        [schema.]column,
        text_query     VARCHAR2
        [,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 string that specifies a query based on the CTXCAT grammar. This query string uses the following tags:

TAG	Description	Possible Values
<query> </query>	Signals that this query be interpreted as a query template.	
<textquery> </textquery>	Specify the query string.	
grammar=	Specify the grammar of the query.	CONTEXT CTXCAT

TAG	Description	Possible Values
<score></score>	Specify the score preference	
datatype=	Specify the type of number returned as score.	INTEGER FLOAT

All tags and attributes values are case-sensitive.

> **See Also:** Chapter 3, "CONTAINS Query Operators" for more information about the operators you can use in query expressions.

label
Optionally specify the label that identifies the score generated by the CONTAINS operator.

Returns

For each row selected, CONTAINS returns a number between 0 and 100 that indicates how relevant the document row is to the query. The number 0 means that Oracle 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 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 always be followed by the > 0 syntax, which specifies that the score value calculated by the CONTAINS operator must be greater than zero for the row to be selected.

When the SCORE operator is called (e.g. 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;
```

Notes

Querying Multi-Language Tables

With the multi-lexer preference, you can create indexes from multi-language tables.

At query time, the multi-lexer examines the session's language setting and uses the sub-lexer 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 document 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;
```

CTXCAT Query Grammar example

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

```
PROMPT
PROMPT create context and ctxcat indexes both with theme indexing on
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
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
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
PROMPT *****  Hitlist from CTXCAT index ***********
select pk||' ==> '||text from test
where catsearch(text,'San Diego','')>0
order by pk;
```

Related Topics

Syntax for CONTEXT Indextype in this chapter

Chapter 3, "CONTAINS Query Operators"

Oracle Text Application Developer's Guide

SCORE

CREATE INDEX

Note: This section describes the CREATE INDEX statement as it pertains to creating a Text domain index.

For a complete description of the CREATE INDEX statement, see *Oracle9i SQL Reference*.

Purpose

Use CREATE INDEX to create an Oracle Text index. An Oracle Text index is an Oracle domain index of type CONTEXT, CTXCAT, CTXRULE or CTXXPATH.

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

You can create the following types of Oracle Text indexes:

- CONTEXT index. This is an index on a text column. You 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 Indextype.
- CTXCAT index. This is a combined index on a text column and one or more other columns.You 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 Indextype.
- CTXRULE index. This is an index on a column containing a set of queries. You query this index with the MATCHES operator in the WHERE clause of a SELECT statement. See Syntax for CTXRULE Indextype.
- CTXXPATH index. Create this index when you need to speed up ExistsNode() queries on an XMLType column. See Syntax for CTXXPATH Indextype.

Required Privileges

You do not need the CTXAPP role to create an Oracle Text index. If you have Oracle grants to create a b-tree index on the text column, you have sufficient permission 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.

Temporary Tablespace Requirements

The creation of an Oracle Text index requires temporary tablespace belonging to the CTXSYS user. Insufficient tablespace results in the ORA-01652 error. To remedy, you extend the CTXSYS tablespace, not the tablespace of the issuing user. Generally, the size of the temporary tablespace required is between 50 and 200 percent of your data.

Syntax for CONTEXT Indextype

Use this indextype to create an index on a text column. You 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(column) INDEXTYPE IS
    ctxsys.context [ONLINE]
    LOCAL [(PARTITION [partition] [PARAMETERS('paramstring')]
        [, PARTITION [partition] [PARAMETERS('paramstring')]])]
    [PARAMETERS(paramstring)] [PARALLEL n] [UNUSABLE];
```

[schema.]index
Specify the name of the Text index to create.

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

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

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

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 indextype, you can create indexes on text and structured columns. See Syntax for CTXCAT Indextype in this chapter.

ONLINE
Creates the index while allowing inserts / updates / deletes (DML) 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.

Limitations The following limitations apply to using ONLINE:

- At the very beginning or very end of this process, DML might fail.

- Local partition index online creation not supported with ONLINE.
- ONLINE is supported for CONTEXT indexes only
- ONLINE cannot be used with PARALLEL

LOCAL [(PARTITION [*partition*] [PARAMETERS('*paramstring*')]
Specify LOCAL to create 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*. If you do not specify a partition name, the system assigns one. The order of the index partition list must correspond to the table partition by order.

The PARAMETERS clause associated with each partition specifies the parameters string specific to that partition. You can only specify *memory* and *storage* for each index partition.

You can 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.

You cannot use the ONLINE parameter with this operation.

> **See Also:** "Creating a Local Partitioned Index"

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 a single partition.

> **See Also:** "Query Performance Limitation with a Partitioned Index" in this chapter under CONTAINS.

PARALLEL n
Optionally specify with n the parallel degree for parallel indexing. The actual degree of parallelism might be smaller depending on your resources.

You can use this parameter on non-partitioned tables. Creating a non-partitioned index in parallel does not turn on parallel query processing.

Using this parameter on a partitioned table results in serial indexing. To index a partitioned table in parallel, you must create an unusable index and then run the `DBMS_PCLXUTIL.BUILD_PART_INDEX` utility.

See Also:

"Parallel Indexing"

"Creating a Local Partitioned Index in Parallel"

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 on a local index turns on parallel queries even though the index is created serially. (Creating a non-partitioned 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 ALTER INDEX NOPARALLEL.

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

Limitations The following limitations apply to using PARALLEL:

- Parallel indexing is supported only for CONTEXT index
- If you specify parallel for local index creation, the parallel clause is ignored, and index creation is done serially. In this case parallel queries are enabled.

  See Also: "Creating a Local Partitioned Index in Parallel" in this section to create a local partitioned index with true parallelism.

- PARALLEL cannot be used with ONLINE.

UNUSABLE

Create an unusable index. This creates index meta data only and exists 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"

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]'
```

You 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 uses the system defaults.

For more information about these defaults, see "Default Index Parameters" in Chapter 2.

---

datastore *datastore_pref*

Specify the name of your datastore preference. Use the datastore preference to specify where your text is stored.See Datastore Types in Chapter 2, "Indexing".

filter *filter_pref*

Specify 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 in Chapter 2, "Indexing".

charset column *charset_column_name*

Specify 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 an NLS character set string such as JA16EUC.

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

You 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.

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

format column *format_column_name*

Specify 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.

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

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

- TEXT
- BINARY
- IGNORE

TEXT indicates that the document is either plain text or HTML. When TEXT is specified the document is not filtered, but might be character set converted.

BINARY indicates that the document is a format supported by the INSO_FILTER object other than plain text or HTML, such as PDF. BINARY is the default if the format column entry cannot be mapped.

IGNORE 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.

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*
Specify the name of your lexer or multi-lexer preference. Use the lexer preference to identify the language of your text and how text is tokenized for indexing. See Lexer Types in Chapter 2, "Indexing".

language column *language_column_name*
Specify the name of the language column when using a multi-lexer preference. See MULTI_LEXER in Chapter 2, "Indexing".

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 is 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*
Specify 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 in Chapter 2, "Indexing".

storage *storage_pref*
Specify 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 in Chapter 2, "Indexing".

stoplist *stoplist*
Specify the name of your stoplist. Use stoplist to identify words that are not to be indexed. See CTX_DDL.CREATE_STOPLIST in Chapter 7, "CTX_DDL Package".

section group *section_group*
Specify the name of your section group. Use section groups to create searchable sections in structured documents. See CTX_DDL.CREATE_SECTION_GROUP in Chapter 7, "CTX_DDL Package".

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

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

where 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.

The memsize parameter specifies the amount of memory Oracle 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 since 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
Specify nopopulate to create an empty index. The default is populate.

> **Note:** This is the only option whose default value cannot be set with CTX_ADM.SET_PARAMETER.

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.

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:** For more information about default settings, see "Default Index Parameters" in Chapter 2.
>
> Also refer to *Oracle Text Application Developer's Guide*.

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 these preferences 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');
```

Creating CONTEXT Index with Multi-Lexer Preference

The multi-lexer decides which lexer to use for each row based on a language column. This is a character column in the table which stores the language of the document in the text column. For example, you 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 multi-lexer preference you created. To index the global_doc table, you specify the multi-lexer 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:** For more information about creating multi-lexer preferences, see MULTI_LEXER in Chapter 2.

Parallel Indexing

This example shows how to set up parallel indexing. Do the following:

Create the index 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

The following example creates a text table 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);
```

Creating a Local Partitioned Index in Parallel

To create a local index in parallel, create an unusable index first, then run the DBMS_PCLXUTIL.BUILD_PART_INDEX utility.

In this example, the base table has three partitions. We create a local partitioned unusable index first, the run the DBMS_PCLUTIL.BUILD_PART_INDEX, which builds the 3 partitions in parallel (inter-partition parallelism). Also inside each partition, index creation is done in parallel (intra-partition parallelism) with a parallel degree of 2.

```
create index tdrbip02bx on tdrbip02b(text)
indextype is ctxsys.context local (partition tdrbip02bx1,
```

```
                                    partition tdrbip02bx2,
                                    partition tdrbip02bx3)
unusable;

exec dbms_pclxutil.build_part_index(3,2,'TDRBIP02B','TDRBIP02BX',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, you can issue:

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

Deleting Index Errors

To clear the index error view, you can issue:

```
DELETE FROM ctx_user_index_errors;
```

Syntax for CTXCAT Indextype

The CTXCAT index is a combined index on a text column and one or more other columns.You 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]
        [wordlist wordlist_pref]
        [memory memsize]');
```

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

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

Supported Preferences

index set *index_set*
Specify the index set preference to create the CTXCAT index. See Creating a CTXCAT Index example in this chapter.

Index Performance and Size Considerations Although a CTXCAT index offers query performance benefits, creating the index has its costs. The time Oracle takes to create a CTXCAT index depends on its total size, and 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 since more indexes must be updated.

Because of these added costs in creating a CTXCAT index, carefully consider the query performance benefit 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 its benefits.

Other Preferences

When you create an index of type CTXCAT, you can use only the following index preferences in the parameters string:

Table 1–1

Preference Class	Supported Types
Datastore	None.
Filter	None
Lexer	BASIC_LEXER (index_themes attribute not supported) CHINESE_VGRAM_LEXER JAPANESE_VGRAM_LEXER KOREAN_LEXER KOREAN_MORPH_LEXER
Wordlist	BASIC_WORDLIST
Storage	BASIC_STORAGE
Stoplist	Supports single language stoplists only (BASIC_STOPLIST type.)
Section Group	None

Unsupported Preferences and Parameters

When you create a CTXCAT index, you cannot specify datastore, filter and section group preferences. You also cannot specify language, format, and 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 the *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 `category_id`. Results must be sorted based on `bid_close` and `category_id`.

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 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','price, bid_close');
end;
```

Create the `CTXCAT` index with `CREATE INDEX` as follows:

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

Querying a CTXCAT Index

To query the title column for the word *pokemon*, you can issue 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 Indextype

This is an index on a column containing a set of queries. You query this index with the MATCHES operator in the WHERE clause of a SELECT statement.

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

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

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

lexer_pref
Specify the lexer preference to be used for processing the queries and the documents to be classified with the MATCHES function. Currently, only the BASIC_LEXER lexer type is supported.

For processing queries, this lexer supports the following operators: ABOUT, STEM, AND, NEAR, NOT, OR, and WITHIN.

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

The following operators are not supported: ACCUM, EQIUV, MINUS, WEIGHT, THRESHOLD, WILDCARD, FUZZY, and SOUNDEX.

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 in Chapter 2, "Indexing".

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 indextype:

- BASIC_SECTION_GROUP
- HTML_SECTION_GROUP

- XML_SECTION_GROUP
- AUTO_SECTION_GROUP

See Section Group Types in Chapter 2, "Indexing".

CTXRULE does not support special sections.

wordlist_pref
Specify the wordlist preferences. This is used to enable stemming operations on query terms.See Wordlist Type in Chapter 2, "Indexing".

Example for Creating a CTXRULE Index

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

Syntax for CTXXPATH Indextype

Create this 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]
              [populate | nopopulate]')];
```

[schema.]table(column)
Specify 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 CTXXPATH.

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 in Chapter 2, "Indexing".

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

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

where 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.

populate | nopopulate
Specify nopopulate to create an empty index. The default is populate.

> **Note:** This is the only option whose default value cannot be set with CTX_ADM.SET_PARAMETER.

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.

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: Oracle9i Application Developer's Guide - XML for information on using the CTXXPATH indextype.

Related Topics

CTX_DDL.CREATE_PREFERENCE in Chapter 7, "CTX_DDL Package".

CTX_DDL.CREATE_STOPLIST in Chapter 7, "CTX_DDL Package".

CTX_DDL.CREATE_SECTION_GROUP in Chapter 7, "CTX_DDL Package".

ALTER INDEX

CATSEARCH

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 *Oracle9i SQL Reference*.

Purpose

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

Syntax

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

[force]
Optionally force the index to be dropped.

Examples

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

```
DROP INDEX doc_index;
```

Notes

Use force option when Oracle cannot determine the state of the index, such as when an indexing operation crashes.

Related Topics

ALTER INDEX

CREATE INDEX

MATCHES

Use this 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.

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

MATCHES returns 1 for one or more matches and 0 for no match.

Limitation

MATCHES does not support functional invocation

Syntax

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

column
Specify the column containing the indexed query set.

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

Example

Assuming that a table querytable has a CTXRULE index associated with it, you can issue the following query that passes in a document string to be classified. The SELECT statement returns all rows (queries) that are satisfied by the incoming document:

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

Related Topics

Syntax for CTXRULE Indextype in this chapter.

Oracle Text Application Developer's Guide

SCORE

Use the SCORE operator in a SELECT statement to return the score values produced by a CONTAINS query.

Syntax

```
SCORE(label NUMBER)
```

label
Specify a number to identify the score produced by the query. You use this number to identify the score in the CONTAINS clause.

Notes

The SCORE operator can be used in a SELECT, ORDER BY, or GROUP BY clause.

Example

Single CONTAINS

When the SCORE operator is called (e.g. 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 NVL(SCORE(10),0), NVL(SCORE(20),0);
```

Related Topics

CONTAINS

Appendix F, "Scoring Algorithm"

2

Indexing

This chapter describes the various elements you can use to create your Oracle Text index.

The following topics are discussed in this chapter:

Overview

When you use CREATE INDEX to create an index or ALTER INDEX 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 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?
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. You 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. You specify `mydatastore` as the datastore preference in the parameter clause of `CREATE INDEX`.

Creating Preferences

To create a datastore, lexer, filter, wordlist, or storage preference, you use the `CTX_DDL.CREATE_PREFERENCE` procedure 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.

To create a stoplists, use `CTX_DDL.CREATE_STOPLIST`. You can 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. You can add sections to section groups with `CTX_DDL. ADD_ZONE_SECTION` or `CTX_DDL.ADD_FIELD_SECTION`.

Datastore Types

Use the datastore types to specify how your text is stored. To create a datastore preference, you must use one of the following 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 per row.
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. Filenames are stored in the text column, one per 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 per row. `DIRECT_DATASTORE` has no attributes.

The following columns types are supported: `CHAR`, `VARCHAR`, `VARCHAR2`, `BLOB`, `CLOB`, `BFILE`, or `XMLType`.

> **Note:** If your column is a `BFILE`, you must grant read permission to `CTXSYS` on all directories used by the `BFILE`s.

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 this datastore when your text is stored in more than one column. During indexing, the system concatenates the text columns and indexes the text as a single document.

MULTI_COLUMN_DATASTORE has the following attributes:

Attribute	Attribute Value
columns	Specify a comma separated list of columns to be concatenated during indexing. You can also specify any expression allowable for the select statement column list for the base table. This includes expressions, PL/SQL functions, column aliases, and so on. 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. RAW and BLOB columns are directly concatenated as binary data. LONG, LONG RAW, NCHAR, and NCLOB, nested table columns and collections are not supported. The column list is limited to 500 bytes.

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 Security

Only CTXSYS is allowed to create preferences for the MULTI_COLUMN_DATASTORE type. Any other user who attempts to create a MULTI_COLUMN_DATASTORE preference receives an error.

Oracle makes this restriction because when the columns attribute contains a function call, the call is made by the CTXSYS schema. The potential exists for a malicious CTXAPP users to execute arbitrary functions for which they do not have execute permission.

If this is too restrictive, you can create a stored procedure under CTXSYS to create MULTI_COLUMN_DATASTORE preferences. The effective user is CTXSYS, who creates and owns the preferences. However, you can call this procedure from any schema as CTXSYS.

MULTI_COLUMN_DATASTORE Example

The following example creates a multi-column 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;
```

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 these tags as sections, you can optionally create field sections with the 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 quotes. For example:

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

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.

DETAIL_DATASTORE has the following attributes:

Attribute	Attribute Value
binary	Specify TRUE for Oracle to add *no* newline character after each detail row. Specify FALSE for Oracle 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(s)
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. You 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

Column Name	Column Type	Description
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

You use CTX_DDL.CREATE_PREFERENCE to create a preference with DETAIL_DATASTORE. You 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, you specify a column in the master table with CREATE INDEX. The column you specify must be one of the allowable types.

This example uses the body column, whose function is to allow 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.

FILE_DATASTORE has the following attribute(s):

Attribute	Attribute Values
path	*path1*:*path2*: :*pathn*

path
Specify 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 only include file names in your text column.

You can specify multiple paths for path, with each path separated by a colon (:). 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, Oracle requires that the path be included in the file names stored in the text column.

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 filenames. 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)

You 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 you specify can be either *ftp*, *http*, or *file*. For example:

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

As 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. Nonprintable ASCII characters and multibyte characters must be escaped with the *%xx* notation, where *xx* is the hexadecimal representation of the special character.

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

URL_DATASTORE Attributes

URL_DATASTORE has the following attributes:

Attribute	Attribute Values
timeout	Specify the timeout in seconds. The valid range is 15 to 3600 seconds. The default is 30.
maxthreads	Specify the maximum number of threads that can be running simultaneously. Use a number between 1and 1024. The default is 8.
urlsize	Specify the maximum length of URL string in bytes. Use **a** number between 32 and 65535. The default is 256.

Attribute	Attribute Values
maxurls	Specify maximum size of URL buffer. Use a number between 32 and 65535. The defaults is 256.
maxdocsize	Specify the maximum document size. Use a number between 256 and 2,147,483,647 bytes (2 gigabytes). The defaults is 2,000,000.
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

Specify the length of time, in seconds, that a network operation such as a connect or read waits before timing out and returning a timeout error to the application. The valid range for timeout is 15 to 3600 and the default is 30.

> **Note:** Since timeout is at the network operation level, the total timeout may be longer than the time specified for timeout.

maxthreads

Specify the maximum number of threads that can be running at the same time. The valid range for maxthreads is 1 to 1024 and the default is 8.

urlsize

Specify the maximum length, in bytes, that the URL data store supports for URLs stored in the database. If a URL is over the maximum length, an error is returned. The valid range for urlsize is 32 to 65535 and the default is 256.

> **Note:** The product values specified for maxurls and urlsize cannot exceed 5,000,000.
>
> In other words, the maximum size of the memory buffer (maxurls * urlsize) for the URL is approximately 5 megabytes.

maxurls
Specify the maximum number of rows that the internal buffer can hold for HTML documents (rows) retrieved from the text table. The valid range for maxurls is 32 to 65535 and the default is 256.

Note: The product values specified for maxurls and urlsize cannot exceed 5,000,000.

In other words, the maximum size of the memory buffer (maxurls * urlsize) for the URL is approximately 5 megabytes.

Specify the maximum size, in bytes, that the URL datastore supports for accessing HTML documents whose URLs are stored in the database. The valid range for maxdocsize is 1 to 2,147,483,647 (2 gigabytes), and the default is 2,000,000.

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 machine 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, separate by commas) which 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 machine(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.oracle.com, uk.oracle.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 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.

The USER_DATASTORE has the following attributes:

Attribute	Attribute Value
procedure	Specify the procedure that synthesizes the document to be indexed. This procedure must be owned by CTXSYS and must be executable by the index owner.
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, since 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. The schema owner name is constrained to CTXSYS, so specifying owner name is not necessary.

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 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:

- procedure must be owned by CTXSYS
- procedure must be executable by the index owner
- procedure cannot issue DDL or transaction control statements like COMMIT

Editing Procedure after Indexing If you change or edit the stored procedure, indexes based upon 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) 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.

Only procedures owned by CTXSYS are allowed for the user datastore. Therefore, CTXSYS must wrap the user procedure (owned by appowner) with a CTXSYS owned procedure as follows:

```
create procedure s_myproc(rid in rowid, tlob in out clob) is
 begin
      appowner.myproc(rid, tlob);
 end;
```

The CTXSYS user must make sure that the index owner can execute the stub procedure by granting execute privileges as follows:

```
grant execute on s_myproc to appowner ;
```

The user appowner creates the preference, setting the procedure attribute to the name of the ctxsys stub procedure as follows:

```
begin
    ctx_ddl.create_preference('myud', 'user_datastore');
    ctx_ddl.set_attribute('myud', 'procedure', 's_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;
```

Because only procedures owned by CTXSYS are allowed for the user datastore, CTXSYS must wrap the user procedure (owned by appowner) with a CTXSYS owned procedure as follows:

```
create procedure s_myproc(rid in rowid, tlob in out blob) is
 begin
      appowner.myds(rid, tlob);
 end;
```

The CTXSYS user must make sure that the index owner can execute the stub procedure by granting execute privileges as follows:

```
grant execute on s_myproc to appowner ;
```

The user appowner creates the preference, setting the procedure and output_type attributes to correspond to the ctxsys stub procedure as follows:

```
begin
    ctx_ddl.create_preference('myud', 'user_datastore');
    ctx_ddl.set_attribute('myud', 'procedure', 's_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.

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 to automatically insert a new line between lines when synthesizing the document text. If you specify TRUE, Oracle 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 checks that the type exists and that the attributes you specify for nested_lineno and nested_text exist in the nested table type. Oracle does not check that the named nested table column exists in the indexed table.

NESTED_DATASTORE Example

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, since 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 allow word processor and formatted documents as well as plain text, HTML, and XML documents to be indexed.

For formatted documents, Oracle stores documents in their native format and uses filters to build temporary plain text or HTML versions of the documents. Oracle indexes the words derived from the plain text or HTML version of the formatted document.

To create a filter preference, you must use one of the following types:

Filter Preference type	Description
CHARSET_FILTER	Character set converting filter
INSO_FILTER	Inso filter for filtering formatted documents
NULL_FILTER	No filtering required. Use for indexing plain text, HTML, or XML documents
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 database character set.

CHARSET_FILTER has the following attribute:

Attribute	Attribute Value
charset	Specify the Globalization Support name of source character set. If you specify UTF16AUTO, 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.

See Also: *Oracle9i Globalization and National Language Support Guide* for more information about the supported Globalization Support character sets.

UTF-16 Big- and Little-Endian Detection

If your character set is UTF-16, you can specify UTF16AUTO to automatically detect big- or little-endian data. Oracle 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 little-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 big-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, you specify the document

character set on a per-row basis. To index the documents, Oracle 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 assumes the source character set is the one specified in the charset attribute.

> **Note:** Character set conversion also works with the INSO_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)
);
```

Insert plain-text documents and name the character set:

```
insert into hdocs values(1, 'text', 'WE8ISO8859P1', '/docs/iso.txt');
insert in 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 ctxsys.charset_filter
  format column fmt
  charset column cset');
```

INSO_FILTER

The INSO_FILTER is a universal filter that filters most document formats. This filtering technology is licensed from Stellent Chicago, Inc.

Use it for indexing single and mixed-format columns.

> **See Also:** For a list of the formats supported by INSO_FILTER and to learn more about how to set up your environment to use this filter, see Appendix B, "Supported Document Formats".

The INSO_FILTER has the following attribute:

Attribute	Attribute Values
timeout	Specify the INSO_FILTER timeout in seconds. Use a number between 0 and 42,949,672. Default is 120. Setting this value 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 INSO_FILTER to bypass the row, such as when format is marked TEXT. Use this feature to prevent the Oracle indexing operation from waiting indefinitely on a hanging Inso filter operation.
timeout_type	Specify either HEURISTIC or FIXED. Default is HEURISTIC. Specify HEURISTIC for Oracle to check every TIMEOUT seconds if output from Output 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 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.

Indexing Formatted Documents

To index a text column containing formatted documents such as Microsoft Word, use the INSO_FILTER. This filter automatically detects the document format. You can use the CTXSYS.INSO_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.inso_filter');
```

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 INSO_FILTER can index mixed-format columns. However, you might want to have the INSO filter bypass the plain text or HTML documents. Filtering plain text or HTML with the INSO_FILTER is redundant.

The format column in the base table allows you to specify the type of document contained in the text column. The only two types you can specify are TEXT and BINARY. During indexing, the INSO_FILTER ignores any document typed TEXT (assuming the charset column is not specified.)

To set up the INSO_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 INSO_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.inso_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 INSO_FILTER.

Character Set Conversion With Inso

The INSO_FILTER converts documents to the database character set when the document format column is set to TEXT. In this case, the INSO_FILTER looks at the charset column to determine the document character set.

If the charset column value is not an Oracle character set name, the document is passed through without any character set conversion.

Note: You need not specify the charset column when using the INSO_FILTER.

If you do specify the charset column and do not specify the format column, the INSO_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 2-31.

Plain Text Indexing and the INSO_FILTER

Oracle does not recommend using INSO_FILTER to index plain text documents.

If your table contains text documents exclusively, use the NULL_FILTER or the USER_FILTER.

If your table contains text documents mixed with formatted documents, Oracle recommends creating a format column and marking the text documents as TEXT to bypass INSO_FILTER. In such cases, Oracle also recommends creating a charset column to indicate the document character set.

However, if you use INSO_FILTER to index nonbinary (text) documents *and* you specify no format column and no charset column, the INSO_FILTER processes the document. Your indexing process is thus subject to the character set limitations of

Inso technology. Specifically, your application must ensure that one of the following conditions is true:

- The document character set is the same as the database character set and the database character set is one of the following:
  - US7ASCII
  - WE8ISO8859P1
  - JA16SJIS
  - KO16KSC5601
  - ZHS16CGB231280
  - ZHT16BIG5
- The database character set is none of those listed and the document is in the WE8ISO8859P1 character set.

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, you can 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" in this chapter.

USER_FILTER

Use the USER_FILTER type to specify an external filter for filtering documents in a column. USER_FILTER has the following attribute:

Attribute	Attribute Values
command	Specify the name of the filter executable.

command

Specify the executable for the single external filter used to filter all text stored in a column. If more than one document format is stored in the column, the external filter specified for command must recognize and handle all such formats.

The executable you specify must exist in the $ORACLE_HOME/ctx/bin directory. You must create your user-filter executable with two parameters: the first is the name of the input file to be read, and the second is the name of the output file to be written to.

If all the document formats are supported by INSO_FILTER, use INSO_FILTER instead of USER_FILTER unless additional tasks besides filtering are required for the documents.

User Filter Example

The following example 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.

This type has the following attributes:

Attribute	Purpose	Allowable Values
procedure	Name of the filter stored procedure.	Any CTXSYS owned 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.

The procedure must be owned by CTXSYS and 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 as passed out 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:

Type	Description
BLOB	The input argument is of type BLOB. The unfiltered document is contained in the BLOB passed in.
CLOB	The input argument is of type CLOB. The unfiltered document is contained in the CLOB passed in. No pre-filtering or character set conversion is done. If the datastore outputs binary data, that binary data is written directly to the CLOB, with Globalization Support doing implicit mapping to character data as best it can.
VARCHAR2	The input argument is of type VARCHAR2. The unfiltered document is contained in the VARCHAR2 passed in. The document can be a maximum of 32767 bytes of data. If the unfiltered document is greater than this length, an error is raised for the document and the filter procedure is not called.

Type	Description
FILE	The input argument is of type VARCHAR2. The unfiltered document content is contained in a temporary file in the file system whose filename is stored in the VARCHAR2 passed in. For example, the value of the passed-in VARCHAR2 might be 'tmp/mydoc.tmp' which means that the document content is stored in the file '/tmp/mydoc.tmp'. The file input type is useful only when your procedure is a safe callout, which can read the file.

The input_type attribute is not mandatory. If not specified, 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, 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.

You 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 via the rowid parameter, but this attribute allows 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_PARAMETER is 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.

You 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_PARAMETER attribute is not mandatory. The default is FALSE.

Parameter Order

ROWID_PARAMETER, FORMAT_PARAMETER, and CHARSET_PARAMETER are all independent. The order is rowid, the format, then charset, but 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, 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)
```

Create Index Requirements

In order to create an index using a PROCEDURE_FILTER preference, the index owner must have execute permission on the procedure.Oracle checks this at index time, which is similar to the security measures for USER_DATASTORE.

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 following 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
CHINESE_VGRAM_LEXER	Lexer for extracting tokens from Chinese text.
CHINESE_LEXER	Lexer for extracting tokens from Chinese text.
JAPANESE_VGRAM_LEXER	Lexer for extracting tokens from Japanese text.
JAPANESE_LEXER	Lexer for extracting tokens from Japanese text.
KOREAN_LEXER	Lexer for extracting tokens from Korean text.
KOREAN_MORPH_LEXER	Lexer for extracting tokens from Korean text (recommended).
USER_LEXER	Lexer you create to index a particular language.

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 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 following attributes:

Attribute	Attribute Values
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)
mixed_case	NO (disabled)

Attribute	Attribute Values
	YES (enabled)
composite	DEFAULT (no composite word indexing, default)
	GERMAN (German composite word indexing)
	DUTCH (Dutch composite word indexing)
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)
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)

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 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
Specify 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 the 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, the character is removed only when it is the last character in the token and it is immediately preceded by the same character.

For example, if the period (.) is defined as both a printjoins and a punctuations character, 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 uses punctuations characters in conjunction with newline and whitespace characters to determine sentence and paragraph deliminters 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, 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

whitespace
Specify 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
Specify 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).

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 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.

In order to retrieve the indexed composite stems, you must issue a stem query, such as *$bahnhof*. The language of the wordlist stemmer must match the language of the composite stems.

Stemming User-Dictionary In your language, you can create a user dictionary to customize how words are decomposed.

You create the user dictionary in the $ORACLE_HOME/ctx/data/<language id> directory. The user dictionary must have the suffix .dct.

For example, the supplied user dictionary file for German is:

```
$ORACLE_HOME/ctx/data/del/drde.dct
```

The format for the user dictionary is as follows:

```
input term <tab> output 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.

If you use the BASIC_LEXER and specify no value for index_themes, this attribute defaults to NO.

You can set this parameter to TRUE for any indextype including CTXCAT. To issue an ABOUT query with CATSEARCH, use the query template with CONTEXT grammar.

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.

You can 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" in Chapter 14, "Executables".

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.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 allows you to query a word in any of its alternate forms.

By default, alternate spelling is enabled in all three languages. You can specify NONE for no alternate spelling.

> **See Also:** For more information about the alternate spelling conventions Oracle uses, see Appendix E, "Alternate Spelling Conventions".

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 printjoins characters set as described, issue 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, you can 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, you specify the language column when you create the index.

You create a multi-lexer preference with the CTX_DDL.CREATE_PREFERENCE. You 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.

Multi-language Stoplists

When you use the MULTI_LEXER, you can also use a multi-language stoplist for indexing.

See Also: Multi-Language Stoplists in this chapter.

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');
```

Since 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 alternate 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. It has no attributes.

You can use this lexer if your database character set is one of the following:

- ZHS16CGB231280

- ZHS16GBK
- ZHT32EUC
- ZHT16BIG5
- ZHT32TRIS
- ZHT16MSWIN950
- ZHT16HKSCS
- UTF8

CHINESE_LEXER

The CHINESE_LEXER type identifies tokens in traditional and simplified Chinese text for creating Text indexes. It has no attributes.

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 new 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.

JAPANESE_VGRAM_LEXER

The JAPANESE_VGRAM_LEXER type identifies tokens in Japanese for creating Text indexes. It has no attributes.

You can use this lexer if your database character set is one of the following:

- JA16SJIS
- JA16EUC
- UTF8

JAPANESE_LEXER

The JAPANESE_LEXER type identifies tokens in Japanese for creating Text indexes. It has no attributes.

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.

The JAPANESE_LEXER supports the following character sets:

- JA16SJIS
- JA16EUC
- UTF8

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)

'自然言語処理'

is indexed as three tokens:

'自然','言語','処理'

In order to resolve a sentence into words, the internal dictionary is referenced. When a word cannot be found in the internal dictionary, Oracle uses the JAPANESE_VGRAM_LEXER to resolve it.

KOREAN_LEXER

The KOREAN_LEXER type identifies tokens in Korean text for creating Text indexes.

> **Note:** This lexer is supported for backward compatibility with older versions of Oracle Text that supported only this Korean lexer. If you are building a new application, Oracle recommends that you use the KOREAN_MORPH_LEXER.

You can use this lexer if your database character set is one of the following:

- KO16KSC5601
- UTF8

When you use the KOREAN_LEXER, you can specify the following boolean attributes:

Attribute	Attribute Values
verb	Specify TRUE or FALSE to index verbs. Default is TRUE.
adjective	Specify TRUE or FALSE to index adjectives. Default is TRUE.
adverb	Specify TRUE or FALSE to index adverb. Default is TRUE.
onechar	Specify TRUE or FALSE to index one character. Default is TRUE.
number	Specify TRUE or FALSE to index number. Default is TRUE.
udic	Specify TRUE or FALSE to index user dictionary. Default is TRUE.
xdic	Specify TRUE or FALSE to index x-user dictionary. Default is TRUE.
composite	Specify TRUE or FALSE to index composite words.
morpheme	Specify TRUE or FALSE for morphological analysis. Default is TRUE.
toupper	Specify TRUE or FALSE to convert English to uppercase. Default is TRUE.
tohangeul	Specify TRUE or FALSE to convert to hanga to hangeul. Default is TRUE.

Limitations

Sentence and paragraph sections are not supported with the Korean lexer.

KOREAN_MORPH_LEXER

The KOREAN_MORPH_LEXER type identifies tokens in Korean text for creating Oracle Text indexes. The KOREAN_MORPH_LEXER lexer offers the following benefits over KOREAN_LEXER:

- better morphological analysis of Korean text
- faster indexing
- smaller indexes
- more accurate query searching

Supplied Dictionaries

The KOREAN_MORPH_LEXER uses four 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 are text format KSC 5601. 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 character set is one of the following:

- KO16KSC5601
- UTF8

Attributes

When you use the KOREAN_MORPH_LEXER, you can specify the following attributes:

Attribute	Attribute Values
verb_adjective	Specify TRUE or FALSE to index verbs and adjectives. 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 example below describes the difference between NGRAM and COMPONENT_WORD.
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 KSC5601 code. Default is FALSE.
english	Specify TRUE to index alphanumeric strings. Default is TRUE.

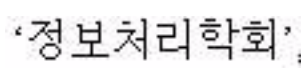
Limitations

Sentence and paragraph sections are not supported with the Korean lexer.

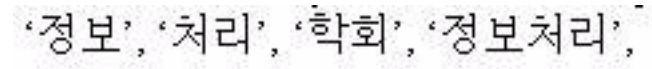
KOREAN_MORPH_LEXER Example: Setting Composite Attribute

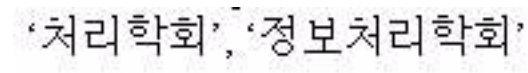
You can use the composite attribute to control how composite nouns are indexed.

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:





You can specify NGRAM indexing as follows:

```
begin
ctx_ddl.create_preference('korean_lexer','KOREAN_MORPH_LEXER');
ctx_ddl.set_attribute('korean_lexer','COMPOSITE','NGRAM');
end
```

To create the index:

```
create index koreanx on korean(text) indextype is ctxsys.context
parameters ('lexer korean_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):

'정보처리학회'

is indexed as four tokens:

'정보처리학회'

'정보', '처리', '학회'

You can specify COMPONENT_WORD indexing as follows:

```
begin
ctx_ddl.create_preference('korean_lexer','KOREAN_MORPH_LEXER');
ctx_ddl.set_attribute('korean_lexer','COMPOSITE','COMPONENT_WORD');
end
```

To create the index:

```
create index koreanx on korean(text) indextype is ctxsys.context
parameters ('lexer korean_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 or to define a new lexer for a language that is supported but inappropriate for your application.

The user-defined lexer you register with Oracle Text is composed of two routines that you must supply:

User-define 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 a 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 indextype
- VGRAM indexing algorithm

USER_LEXER Attributes

The USER_LEXER has the following attributes:

Attribute	Supported Values
INDEX_PROCEDURE	Name of a stored procedure. No default provided.
INPUT_TYPE	VARCHAR2, CLOB. Default is CLOB.
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.

This stored procedure must be owned by user CTXSYS and the index owner must have EXECUTE privilege on it.

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
- issue 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 allows 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 Table 2–1 describes the interface that allows 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 2–2 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 2–1 VARCHAR2 Interface for INDEX_PROCEDURE

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.

Parameter Position	Parameter Mode	Parameter Datatype	Description
3	IN	BOOLEAN	Is Location information needed? Oracle Text sets this parameter to TRUE when 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 2–2 describes the CLOB interface that allows 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 2–2 CLOB interface for INDEX_PROCEDURE

Parameter Position	Parameter Mode	Parameter Datatype	Description
1	IN	CLOB	Same as Table 2–1, "VARCHAR2 Interface for INDEX_PROCEDURE".
2	IN OUT	CLOB	Same as Table 2–1, "VARCHAR2 Interface for INDEX_PROCEDURE". The IN value will always be a truncated CLOB.
3	IN	BOOLEAN	Same as Table 2–1, "VARCHAR2 Interface for INDEX_PROCEDURE".

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 *words* in the query. A space-delimited group of characters (excluding the query operators) in the query will be identified by Text as a *word*.

Requirements This procedure can be a PL/SQL stored procedure.

This stored procedure must be owned by user CTXSYS and the index owner must have EXECUTE privilege on it.

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
- issue 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 2–3 describes the interface for the user-defined lexer query procedure:

Table 2–3

Parameter Position	Parameter Mode	Parameter Datatype	Description
1	IN	VARCHAR2	Query *word* to be tokenized.

Parameter Position	Parameter Mode	Parameter Datatype	Description
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.
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.

- User-defined Indexing Procedure without Location XML Schema
- User-defined Indexing Procedure with Location XML Schema
- 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 (i.e. <!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 (i.e. <?xml version="1.0" ...?>).
- Namespaces.
- Use of elements and attributes other than those defined by the corresponding XML Schema.
- Character references (e.g. ট).
- xml:space attribute.
- xml:lang attribute

User-defined Indexing Procedure without Location XML Schema

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: 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 2–4 describes the element names defined in the preceding XML Schema.

Table 2–4 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.

Table 2–4 Element names

Element	Description
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	This element implies that the previous word token in the sequence was a composite token and that this compMem token is a constituent of that 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> HAUPT </compMem>
  <compMem> HAUPTBAHNHOF </compMem>
  <compMem> NORD </compMem>
  <word> UND </word>
  <word> AUS </word>
  <word> DER </word>
  <word> INNENSTADT </word>
  <word> ZUM </word>
  <word> MESSEGELÄNDE </word>
  <eos/>
</tokens>
```

Example Document: Oracle9i Release 2

Tokens:

```
<tokens>
  <word> ORACLE9I</word>
  <word> RELEASE </word>
```

```
  <num> 2 </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>
```

User-defined Indexing Procedure with Location XML Schema

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 w.r.t 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"/>
    </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: 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 2–4, "Element names" describes the element types defined in the preceding XML Schema.

Table 2–5, "Attributes" describes the attributes defined in the preceding XML Schema.

Table 2–5 Attributes

Attribute	Description
off	This attribute represents the character offset (same semantics as SQL function LENGTH) 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 and / or the section group object before being passed to the user-defined lexer indexing procedure. The offset of the first character in the document being tokenized is 0 (zero).
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.

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>
```

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"/>
    </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: 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 num, word, and compMem 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 2–4, "Element names" describes the element types defined in the preceding XML Schema.

Table 2–6, "Attribute for XML Schema: Query Procedure" describes the attribute defined in the preceding XML Schema.

Table 2–6 Attribute for XML Schema: Query Procedure

Attribute	Description
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. 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>
```

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:** For more information about the stem and fuzzy operators, see Chapter 3, "CONTAINS Query Operators".

BASIC_WORDLIST has the following attributes:

Table 2–7

Attribute	Attribute Values
stemmer	Specify which language stemmer to use. You can specify one of the following: NULL (no stemming) ENGLISH (English inflectional) DERIVATIONAL (English derivational) DUTCH FRENCH GERMAN ITALIAN SPANISH AUTO (automatic language-detection for stemming)

Table 2–7

Attribute	Attribute Values
fuzzy_match	Specify which fuzzy matching cluster to use. You can specify one of the following: GENERIC JAPANESE_VGRAM KOREAN CHINESE_VGRAM ENGLISH DUTCH FRENCH GERMAN ITALIAN SPANISH OCR AUTO (automatic language detection for stemming)
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 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%. Defaults to FALSE.
prefix_length_min	Specify the minimum length of indexed prefixes. Defaults to 1.
prefix_length_max	Specify the maximum length of indexed prefixes. Defaults to 64.
wlidcard_maxterms	Specify the maximum number of terms in a wildcard expansion. Use a number between 1 and 15,000. Default is 5,000.

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 session. When there is no stemmer for a language, the default is NULL. With the NULL stemmer, the stem operator is ignored in queries.

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 attribute values 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 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 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

- The size of the substring index created is approximately the size of the $X index on the word table.
- 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%*. Defaults to `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 like this in the $I table:

Token	Type	Information
TOKEN	0	DOCID 1 POS 1
TOY	0	DOCID 1 POS 3

With prefix indexing, Oracle 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 does no expansion of terms and merging of result sets. To obtain the result, Oracle need only examine the (TO,6) row.

prefix_length_min
Specify the minimum length of indexed prefixes. Defaults to 1.

For example, setting `prefix_length_min` to 3 and `prefix_length_max` 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_length_max
Specify the maximum length of indexed prefixes. Defaults to 64.

For example, setting `prefix_length_min` to 3 and `prefix_length_max` 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 returns an error when the wildcard query expansion exceeds this number.

BASIC_WORDLIST Example

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, issue 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 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:

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 *Oracle9i SQL Reference*.

BASIC_STORAGE has the following attributes:

Attribute	Attribute Value
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.
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)'

Attribute	Attribute Value
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.
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 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, since such compression saves disk space and helps query performance.
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.

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, issued 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 Example

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', 'N_TABLE_CLAUSE',
```

```
                        'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'I_INDEX_CLAUSE',
                        'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'P_TABLE_CLAUSE',
                        'tablespace foo storage (initial 1K)');
end;
```

Section Group Types

In order to issue WITHIN queries on document sections, you must create a section group before you define your sections. You 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:

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 characters. 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

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 CTX_DDL.ADD_SECTION procedure. To index your documents, you can issue a statement such as:

```
create index myindex on docs(htmlfile) indextype is ctxsys.context
parameters('filter ctxsys.null_filter section group htmgroup');
```

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 CTX_DDL.ADD_SECTION procedure. To index your documents, you can issue a statement such as:

```
create index myindex on docs(htmlfile) indextype is ctxsys.context
parameters('filter ctxsys.null_filter section group xmlgroup');
```

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 type used to create a preference for CTX_CLS.TRAIN.

RULE_CLASSIFIER

Use the RULE_CLASSIFIER type for creating preferences for the rule generating procedure, CTX_CLS.TRAIN.

This type has the following attributes:

Attribute Name	Data Type	Default	Min Value	Max Value	Description
THRESHOLD	I	50	1	99	Threshold (in percentage) from 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. This attribute specifies the maximum number of terms that can be selected for each class.
MEMORY_SIZE	I	500	10	4000	Typical memory usage for training in MB. Larger values improve performance.
NT_THRESHOLD	F	0.001	0	0.90	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.

Attribute Name	Data Type	Default	Min Value	Max Value	Description
TERM_ THRESHOLD	I	10	0	100	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.

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 languages including English, French, German, Spanish, Dutch, and Danish. These default stoplists contain only stopwords.

> **See Also:** For more information about the supplied default stoplists, see Appendix D, "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, and Japanese.

To create a multi-language stoplist, use the `CTX_DLL.CREATE_STOPLIST` procedure and specify a stoplist type of `MULTI_STOPLIST`. You 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

You can 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`.

Modifying the Default Stoplist

The default stoplist is always named `CTXSYS.DEFAULT_STOPLIST`. You can 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, "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 2-93.

System-defined preferences are divided into the following categories:

- Data Storage
- Filter
- Lexer
- Section Group
- Stoplist
- Storage
- Wordlist

Data Storage

CTXSYS.DEFAULT_DATASTORE

This preference uses the DIRECT_DATASTORE type. You can 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

CTXSYS.NULL_FILTER

This preference uses the NULL_FILTER type.

CTXSYS.INSO_FILTER

This preference uses the INSO_FILTER type.

Lexer

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, 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 2-38.

CTXSYS.BASIC_LEXER

This preference uses the BASIC_LEXER.

Section Group

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

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 D, "Supplied Stoplists".

CTXSYS.EMPTY_STOPLIST

This stoplist has no words.

Storage

CTXSYS.DEFAULT_STORAGE

This storage preference uses the BASIC_STORAGE type.

Wordlist

CTXSYS.DEFAULT_WORDLIST

This preference uses the language stemmer for your database language. If your language is not listed in Table 2–7 on page 2-71, this preference defaults to the NULL stemmer and the GENERIC fuzzy matching attribute.

System Parameters

This section describes the Oracle Text system parameters. They fall 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:

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.
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 8-22.

You can view system defaults by querying the CTX_PARAMETERS view. You can change defaults using the CTX_ADM.SET_PARAMETER procedure.

Default Index Parameters

This section describes the index parameters you can use when you create context and ctxcat indexes.

CONTEXT Index Parameters

The following default parameters are used when you do not specify preferences in the parameter clause of CREATE INDEX when you create a context index. Each default parameter names a system-defined preference to use for data storage, filtering, lexing, and so on.

System 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 ■ You specify a datastore preference that uses FILE_DATASTORE	CTXSYS.INSO_FILTER
DEFAULT_FILTER_BINARY	No filter preference specified in parameter clause of CREATE INDEX, and Oracle detects that the text column datatype is RAW, LONG RAW, or BLOB.	CTXSYS.INSO_FILTER
DEFAULT_FILTER_TEXT	No filter preference specified in parameter clause of CREATE INDEX, and Oracle 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 INSO_FILTER.	CTXSYS.HTML_SECTION_GROUP

System Parameter	Used When	Default Value
DEFAULT_SECTION_TEXT	No section group specified in parameter clause of CREATE INDEX, and when you do *not* use either URL_DATASTORE or INSO_FILTER.	CTXSYS.NULL_SECTION_GROUP
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.

System Parameter	Used When	Default Value
DEFAULT_CTXCAT_INDEX_SET	No index set specified in parameter clause of CREATE INDEX.	
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

System Parameter	Used When	Default Value
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, since 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

The following default parameters 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.

System 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

Viewing Default Values

You can view system defaults by querying the CTX_PARAMETERS view. For example, to see all parameters and values, you can issue:

```
SQL> SELECT par_name, par_value from ctx_parameters;
```

Changing Default Values

You can change a default value using the CTX_ADM.SET_PARAMETER procedure to name another custom or system-defined preference to use as default.

3

CONTAINS Query Operators

This chapter describes operator precedence and provides description, syntax, and examples for every CONTAINS operator. The following topics are covered:

- 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 below 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. WITHIN
7. AND (&)
8. OR (|)
9. 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. ABOUT
3. stem ($)
4. Fuzzy
5. 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

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 brackets, expansion operators are not applied to expressions unless the operators are also within the brackets

See Also: Grouping Characters in Chapter 4, "Special Characters in Queries".

ABOUT

General Behavior

In all languages, an ABOUT query increases the number of relevant documents returned from the same query without this operator. Oracle 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.

Note: You need not have a theme component in the index to issue ABOUT queries in English. However, having a theme component in the index yields the best results for ABOUT queries.

Oracle retrieves documents that contain concepts that are related to your query word or phrase. For example, if you issue 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 normalizes the word or phrase before performing lookup in the index.

You can use the ABOUT operator with the CONTAINS and CATSEARCH SQL operators.

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.

You can 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, "Executables".

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, returns documents that contain concepts related to *phrase*. The score returned is a relevance score. Oracle 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 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.

Limitations

- The phrase you specify in an ABOUT query cannot be more than 4000 characters.
- You cannot combine the WITHIN operator with ABOUT operator like '*ABOUT (xyz) WITHIN abc*'.
- You cannot combine ABOUT with any operator involving offset information, such as NEAR or WITHIN.

Examples

Single Words

To search for documents that are about soccer, use the following syntax:

```
'about(soccer)'
```

Phrases

You can 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 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

You can use other operators, such as AND or NOT, to combine ABOUT queries with word queries.

For example, you can issue the following combined ABOUT and word query:

```
'about(dogs) and cat'
```

You can combine an ABOUT query with another ABOUT query as follows:

```
'about(dogs) not about(labradors)'
```

> **Note:** You cannot combine ABOUT with the WITHIN operator like '*ABOUT (xyz) WITHIN abc*'.

ABOUT Query with CATSEARCH

You can issue 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 of the query terms. The accumulate operator ranks documents according to the total term weight of a document.

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.

Examples

The following example returns documents that contain either *soccer*, *Brazil*, or *cup* and assigns the highest scores to the documents that contain all three terms:

```
'soccer, Brazil, cup'
```

The following example also returns documents that contain either *soccer*, *Brazil*, or *cup*. However, the weight operator ensures that documents with *Brazil* score higher than documents that contain only *soccer* and *cup*.

```
'soccer, Brazil*3, cup'
```

Notes

Accumulate Scoring

ACCUM scores documents based on two criteria:

- document term weights
- document term scores

Term weight refers to the weight you place on a query term. A query such as x,y,z has term weights of 1 for each term. A query of x, 3*y, z, has term weights of 1, 3, and 1 for the individual terms.

Accumulate scoring guarantees that if a document A matches *p* terms with a total term weight of *m*, and document B matches *q* terms with a total term weight of *m*+1,

document B is guaranteed to have a higher relevance score than document A, regardless of the numbers *p* and *q*.

If two documents have the same weight M, the higher relevance score goes to the document with the higher weighted average term score.

This following table illustrates accumulate scoring:

Document	query	Score(x)	Score(y)	Score(z)	Total Term Weight	Score(query)
A	x,y,z	10	0	0	1	3
B	x,y,z	10	20	0	2	38
C	x,y,z	10	20	30	3	73
D	x,y,z	50	50	0	2	50
E	x, y*3, z	100	0	100	2	40
F	x, y*3, z	0	1	0	3	41

Each row in the table shows the score for an accumulate query. The first four rows show the scores for query x,y,z for documents A, B, C, D. The next two rows show the scores for query x, y*3,z for documents E and F. Assume that x, y and z stand for three different words. The query for document E and F has a weight of 3 on the second query term to arbitrarily make it the most important query term.

The total document term weight is shown for each document. For example, document A has a matching weight of one since only one query term matches the document. Similarly document C has a weight of 3 since all query terms with weight 1 match the document.

The table shows that documents that have higher query term weights are always scored higher than those that contain lower query term weights. For example, document C always scores higher than documents A, B, and D, since document C has the highest query term weight. Similarly, document F scores higher than document E, since F has a higher matching weight.

For documents that have equal term weights, such as document B and D, the higher score goes to the document with the higher weighted average term score, which is document D.

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.

Examples

To obtain all the documents that contain the terms *blue* and *black* and *red*, issue 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.

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 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, 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

You can browse a thesaurus using procedures in the CTX_THES package.

See Also: For more information on browsing the broader terms in your thesaurus, see CTX_THES.BT in Chapter 12, "CTX_THES Package".

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.

Examples

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 new 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, German, Italian, Dutch, Spanish, and OCR.

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 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 expands *term* to include words only in the index.
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.

Parameter	Description
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.

You can 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.

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.

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 top-level tags <A> and </A>. The A tag must be a top-level tag, which is the document-type tag.

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

You can 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. Instead use

```
dog INPATH (a)
```

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 allowed.
- 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 BLACK GOLD">dog</A>
  ```
  because OF is a default stopword in English and the query matches any word in that position.

  ```
  <A B="POT_OF_GOLD">dog</A>
  ```
  because the underscore character is not a join character by default.

MINUS (-)

Use the MINUS operator to search for documents that contain one query term and you want the presence of a second query term to cause the document to be ranked lower. The MINUS operator is useful for lowering the score of documents that contain unwanted noise terms.

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.

Examples

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*.

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, 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, 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 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 only expands the query using the branch corresponding to the specified narrower term operator.

Related Topics

You can browse a thesaurus using procedures in the CTX_THES package.

See Also: For more information on browsing the narrower terms in your thesaurus, see CTX_THES.NT in Chapter 12, "CTX_THES Package".

NEAR (;)

Use the NEAR operator to return a score based on the proximity of two or more query terms. Oracle 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]])

word*1-n*
Specify the terms in the query separated by commas. The query terms can be single words or phrases.

max_span
Optionally specify the size of the biggest clump. The default is 100. Oracle 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*, issue the following query:

```
'near((dog, cat), 6)'
```

order
Specify TRUE for Oracle 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, issue the following query:

```
'near((monday, tuesday, wednesday), 20, TRUE)'
```

Note: To specify order, you must always specify a number for the max_span parameter.

Oracle might return different scores for the same document when you use identical query expressions that have the order flag set differently. For example, Oracle might return different scores for the same document when you issue 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 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, since 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. You can define clump size with the max_span parameter as described in this section.

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, issue 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*.

Backward Compatibility NEAR Syntax

You can write near queries using the syntax of previous ConText releases. For example, to find all documents where *lion* occurs near *tiger*, you can 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
carmel 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:** For more information about the procedures you can use for highlighting, see Chapter 8, "CTX_DOC Package".

Section Searching and NEAR

You can 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 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.

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

For example, 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.

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.

Examples

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

You can browse a thesaurus using procedures in the CTX_THES package.

> **See Also:** For more information on browsing the preferred terms in your thesaurus, see CTX_THES.PT in Chapter 12, "CTX_THES Package".

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.

Examples

The term *dog* has a related term of *wolf*. A RT query for *dog* returns all documents that contain the word *dog* and *wolf*.

Related Topics

You can browse a thesaurus using procedures in the CTX_THES package

See Also: For more information on browsing the related terms in your thesaurus, see CTX_THES.RT in Chapter 12, "CTX_THES Package".

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 allows 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).

Examples

```
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, Oracle 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.

Stemming performance can be improved by using the index_stems attribute of the BASIC_LEXER preference.

The Oracle Text stemmer, licensed from Xerox Corporation's XSoft Division, supports the following languages: English, French, Spanish, Italian, German, and Dutch.

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`.

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);
```

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

You can browse your thesaurus using procedures in the CTX_THES package.

> **See Also:** For more information on browsing the synonym terms in your thesaurus, see CTX_THES.SYN in Chapter 12, "CTX_THES Package".

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. 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, 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*, issue the following query:

```
'tr(cat, spanish, my_thes)'
```

This query expands to:

```
'{cat}|{gato}|{chat}'
```

Related Topics

You can browse a thesaurus using procedures in the CTX_THES package.

> **See Also:** For more information on browsing the related terms in your thesaurus, see CTX_THES.TR in Chapter 12, "CTX_THES Package".

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, 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*, issue the following query:

```
'trsyn(cat, spanish, my_thes)'
```

This query expands to:

```
'{cat}|{gato}|{lion}|{leon}'
```

Related Topics

You can browse a thesaurus using procedures in the CTX_THES package.

> **See Also:** For more information on browsing the translation and synonym terms in your thesaurus, see CTX_THES.TRSYN in Chapter 12, "CTX_THES Package".

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 (BT, NT) in a thesaurus. 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.

Examples

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

You can browse your thesaurus using procedures in the CTX_THES package.

See Also: For more information on browsing the top terms in your thesaurus, see CTX_THES.TT in Chapter 12, "CTX_THES Package".

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. You can reduce the score of a query term by using the weight operator with a number less than 1; you can 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, OR, or AND queries when the expression has more than one query term. With no weighting on individual terms, the score cannot tell you 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

You have a collection of sports articles. You are interested in the articles about soccer, in particular 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, you can issue the following query:

```
'soccer or Brazil*3'
```

Table 3–1 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 3–1

	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.

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 specifies that any characters can appear in multiple positions represented by the wildcard.
_	The underscore wildcard specifies a single position in which any character can occur.

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*, you can write your query as follows:

```
'_ing'
```

You can write this query more generally as:

```
'%ing'
```

You can also 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

You can 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:** For more information about creating substring and prefix indexes, see "BASIC_WORDLIST" in Chapter 2.

WITHIN

You can 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.

Examples

Querying Within Zone Sections

To find all the documents that contain the term *San Francisco* within the section *Headings*, write your 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 your 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 your 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 your 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. You 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: You can make text within a field section invisible.
- Repeatability: WITHIN queries cannot distinguish repeated field sections.
- Nestability: You cannot issue 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, you can use the WITHIN operator as follows:

```
'dog WITHIN TITLE'
```

Alternatively, you can set the visible flag to TRUE when you define TITLE as a field section with CTX_DDL.ADD_FIELD_SECTION.

> **See Also:** For more information about creating field sections, see ADD_FIELD_SECTION in Chapter 7, "CTX_DDL Package".

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 issue 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. You 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

You can query within attribute sections when you index with either XML_SECTION_GROUP or AUTOMATIC_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>
```

You can define the section `title@book` to be the attribute section `title`. You can 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`, you issue 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 meta-data 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.

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*
- You cannot combine `WITHIN` with expansion operators, such as $! and *.
- Since `WITHIN` is a reserved word, you must escape the word with braces to search on it.
- You cannot combine the WITHIN operator with ABOUT operator like 'A*BOUT (xyz) WITHIN abc*'.

4

Special Characters in 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
- Escape Characters
- Reserved Words and Characters

Grouping Characters

The grouping characters control operator precedence by grouping query terms and operators in a query expression. The grouping characters are:

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:

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.

In the following examples, an escape sequence is necessary because each expression contains a Text operator or reserved symbol:

```
'AT\&T'
'{AT&T}'

'high\-voltage'
'{high-voltage}'
```

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

The following table lists the Oracle Text reserved words and characters that must be escaped when you want to search them in `CONTAINS` queries:

Reserved Word	Reserved Character	Operator
ABOUT	(none)	ABOUT
ACCUM	,	Accumulate
AND	&	And
BT	(none)	Broader Term
BTG	(none)	Broader Term Generic
BTI	(none)	Broader Term Instance
BTP	(none)	Broader Term Partitive
FUZZY	?	fuzzy
(none)	{ }	escape characters (multiple)
(none)	\	escape character (single)
(none)	( )	grouping characters
(none)	[ ]	grouping characters
HASPATH	(none)	HASPATH
INPATH	(none)	INPATH
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

Reserved Word	Reserved Character	Operator
(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)
WITHIN	(none)	WITHIN

5

CTX_ADM Package

This chapter provides reference information for using the CTX_ADM PL/SQL package to administer servers and the data dictionary.

CTX_ADM contains the following stored procedures:

Name	Description
RECOVER	Cleans up database objects for deleted Text tables.
SET_PARAMETER	Sets system-level defaults for index creation.
SHUTDOWN	Shuts down a single ctxsrv server or all currently running servers.

Note: Only the CTXSYS user can use the procedures in CTX_ADM.

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;
```

Notes

You need not call CTX_ADM.RECOVER to perform system recovery if ctxsrv servers are running; any *ctxsrv* servers that are running automatically perform system recovery approximately every fifteen minutes. RECOVER provides a method for users to perform recovery on command.

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:

- 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_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" in Chapter 2.

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[M|G|K]
```

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

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;
```

SHUTDOWN

The SHUTDOWN procedure shuts down the specified *ctxsrv* server.

Syntax

```
CTX_ADM.SHUTDOWN(name   IN VARCHAR2 DEFAULT 'ALL',
                 sdmode IN NUMBER   DEFAULT NULL);
```

name
Specify the name (internal identifier) of the *ctxsrv* server to shutdown.

Default is ALL.

sdmode
Specify the shutdown mode for the server:

- 0 or NULL (Normal)
- 1 (Immediate)
- 2 (Abort)

Default is NULL.

Examples

```
begin
ctx_adm.shutdown('DRSRV_3321', 1);
end;
```

Notes

If you do not specify a ctxsrv server to shut down, CTX_ADM.SHUTDOWN shuts down all currently running ctxsrv servers.

The names of all currently running ctxsrv servers can be obtained from the CTX_SERVERS view.

Related Topics

Thesaurus Loader (ctxload) in Chapter 14, "Executables"

6

CTX_CLS Package

This chapter provides reference information for using the CTX_CLS PL/SQL package to generate CTXRULE rules for a set of documents.

Name	Description
TRAIN	Generates rules that define document categories. Output based on input training document set.

TRAIN

Use this procedure to generate query rules that select document categories. You must supply a training set consisting of categorized documents. Each document must belong to one or more categories. This procedure generates the queries that define the categories and then writes the results to a table.

This procedure requires that your document table have an associated populated context index. For best results, the index should be synchronized before running this procedure.

You must also have a document table and a category table. The documents can be in any format supported by Oracle Text.

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 CONSTRAINT fk_id REFERENCES trainingdoc(docid),
    categoryid number);
```

Syntax

```
CTX_CLS.TRAIN(
    index_name in varchar2,
    doc_id in varchar2,
    cattab in varchar2,
    catdocid in varchar2,
    catid in varchar2,
    restab in varchar2,
    rescatid in varchar2,
    resquery in varchar2,
    resconfid in varchar2,
    preference_name in varchar2 DEFAULT NULL
);
```

index_name
Specify the name of the context index associated with your document training set.

doc_id
Specify the name of the document id column in the document table. This column must contain unique document ids. This column must a NUMBER.

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.

catid
Specify the name of the category ID column in the category table. This column must a NUMBER.

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.

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)'

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_name
Specify the name of the preference. For attributes, see "Classifier Types" in Chapter 2, "Indexing".

Example

The CTX_CLS.TRAIN procedure requires that your document table have an associated context index. For example your document table can be defined and populated as follows:

```
set serverout on
exec dbms_output.put_line(TO_CHAR(SYSDATE,'MM-DD-YYYY HH24:MI:SS')||':start');

create table doc (id number primary key, text varchar2(2000));
insert into doc values(1,'In 2002, Europe changed its currency to the EURO');
insert into doc values(2,'The NASDAQ rose today in heavy stock trading.');
insert into doc values(3,'The EURO lost 1 cent today against the US dollar');
insert into doc values(4,'Salt Lake City hosts the winter Olympic games');
insert into doc values(5,'ESPN broadcasts World Cup Soccer games.');
insert into doc values(6,'Soccer champion Diego Maradona retires.');
```

Create the CONTEXT index:

```
exec ctx_ddl.drop_preference('my_lexer');
exec ctx_ddl.create_preference('my_lexer','BASIC_LEXER');
exec ctx_ddl.set_attribute('my_lexer','INDEX_THEMES','NO');
exec ctx_ddl.set_attribute('my_lexer','INDEX_TEXT','YES');
CREATE INDEX docx on doc(text) INDEXTYPE IS ctxsys.context
PARAMETERS('LEXER my_lexer');
```

You must also create a category table as follows to relate the documents to categories:

```
create table category (doc_id number, cat_id number, cat_name varchar2(100));
insert into category values (1,1,'Finance');
insert into category values (2,1,'Finance');
insert into category values (3,1,'Finance');
insert into category values (4,2,'Sports');
insert into category values (5,2,'Sports');
insert into category values (6,2,'Sports');
```

CTX_CLS.TRAIN writes to result table that can be defined like:

```
create table restab (cat_id number, query VARCHAR2(400), conf number);
```

To populate the result table for later CTXRULE indexing, set your RULE_CLASSIFIER preference attributes and call CTX_CLS.TRAIN as follows:

```
exec ctx_ddl.drop_preference('my_classifier');
exec ctx_ddl.create_preference('my_classifier','RULE_CLASSIFIER');
exec ctx_ddl.set_attribute('my_classifier','MAX_TERMS','20');
exec ctx_ddl.set_attribute('my_classifier','THRESHOLD','40');
exec ctx_ddl.set_attribute('my_classifier','NT_THRESHOLD','0.02');
exec ctx_ddl.set_attribute('my_classifier','MEMORY_SIZE','200');
exec ctx_ddl.set_attribute('my_classifier','TERM_THRESHOLD','20');
exec ctx_output.start_log('mylog');

exec
ctx_cls.train('docx','id','category','doc_id','cat_id','restab','cat_
id','query', 'conf','my_classifier');
exec ctx_output.end_log();

create table catname as (select distinct cat_id, cat_name from category);

set termout on
select rpad(id,6) doc_id , rpad(cat_name,8) cat_name, rpad(text,50) text
 from doc, category where id=doc_id;
select rpad(a.cat_id,8) cat_id, rpad(cat_name,8) cat_name,  rpad(query,30) rule
 from restab a, catname b where b.cat_id=a.cat_id;
```

The training set is:

```
DOC_ID CAT_NAME TEXT
------ -------- --------------------------------------------------
1      Finance  In 2002, Europe changed its currency to the EURO
2      Finance  The NASDAQ rose today in heavy stock trading.
3      Finance  The EURO lost 1 cent today against the US dollar
4      Sports   Salt Lake City hosts the winter Olympic games
5      Sports   ESPN broadcasts World Cup Soccer games.
6      Sports   Soccer champion Diego Maradona retires.

6 rows selected.
```

The generated rules for the categories of FINANCE and SPORTS are as follows:

```
CAT_ID   CAT_NAME RULE
-------- -------- ------------------------------
1        Finance  EURO
1        Finance  TODAY ~ EURO
```

```
2        Sports   GAMES
2        Sports   SOCCER ~ GAMES
```

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 a section group.
ADD_FIELD_SECTION	Creates a filed section and assigns it to the specified section group
ADD_INDEX	Adds an index to a catalog index preference.
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.
CREATE_INDEX_SET	Creates an index set for CTXCAT index types.
CREATE_POLICY	Create 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_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_STOPLIST	Drops a stoplist.
OPTIMIZE_INDEX	Optimize the index.
REMOVE_INDEX	Removes an index from a CTXCAT index preference.
REMOVE_SECTION	Deletes a section from a section group
REMOVE_STOPCLASS	Deletes a stopclass from a section group.
REMOVE_STOPTHEME	Deletes a stoptheme from a stoplist.
REMOVE_STOPWORD	Deletes a stopword from a section group.
SET_ATTRIBUTE	Sets a preference attribute.
SYNC_INDEX	Synchronize 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 allows 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 email- 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. You use this name to identify the section in queries. Avoid using names that contain non-alphanumeric characters such as _, since 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, making details transparent to searches.

tag
Specify the tag which 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.

If group_name is an HTML_SECTION_GROUP, 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.

Oracle 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 rest of document.

By default the visible flag is FALSE. This means that Oracle 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 code 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. If you want 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, you can 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 operator in Chapter 3, "CONTAINS Query Operators".

"Section Group Types" in Chapter 2, "Indexing".

CREATE_SECTION_GROUP

ADD_ZONE_SECTION

ADD_SPECIAL_SECTION

REMOVE_SECTION

DROP_SECTION_GROUP

ADD_INDEX

Use this procedure to add an index to a catalog index preference. You create this preference to create catalog indexes of type CTXCAT.

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.

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');
ctx_ddl.add_index('auction_iset','price, 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*, you can issue 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;
```

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 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

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 code 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;
```

You can also 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, you can 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 operator in Chapter 3, "CONTAINS Query Operators".

"Section Group Types" in Chapter 2, "Indexing".

CREATE_SECTION_GROUP

ADD_ZONE_SECTION

ADD_FIELD_SECTION

REMOVE_SECTION

DROP_SECTION_GROUP

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.

The maximum number of stopwords, stopthemes, and stopclasses you can add to a stoplist is 4095.

Example

The following code adds a stopclass of NUMBERS to the stoplist mystop:

```
begin
ctx_ddl.add_stopclass('mystop', 'NUMBERS');
end;
```

Related Topics

CREATE_STOPLIST

REMOVE_STOPCLASS

DROP_STOPLIST

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, 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.

You can 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 code 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 code also stops any attribute sections contained within <fluff>. For example, if a document contained:

```
<fluff type="computer">
```

Then the above code also stops the attribute section fluff@type.

Doctype Sensitive Stop Sections

The following code 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 in Chapter 1, "SQL Statements and Operators".

CREATE_SECTION_GROUP

ADD_STOPTHEME

Adds a single stoptheme to a stoplist. A stoptheme is a theme that is not to be indexed.

In English, you 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, 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

REMOVE_STOPTHEME

DROP_STOPLIST

ABOUT operator in Chapter 3, "CONTAINS Query Operators".

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-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: You can 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

REMOVE_STOPWORD

DROP_STOPLIST

ALTER INDEX in Chapter 1, "SQL Statements and Operators".

Appendix D, "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, you can specify ENGLISH or EN for English.

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, you cannot specify an alt_value.

The alt_value is limited to 30 bytes and cannot be an 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 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_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');
```

Since 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 we add those as alternate 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. You use this name to identify the section in WITHIN queries. Avoid using names that contain non-alphanumeric characters such as _, since 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 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 code 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, you can 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, you can 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. You want to create a distinct book section for each document type.

Assume that mydocname is declared as an XML document type (root element) as follows:

```
<!DOCTYPE mydocname ... [...
```

Within mydocname, the element <book> is declared. For this tag, you can 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', '(mydocname)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 operator in Chapter 3, "CONTAINS Query Operators".

"Section Group Types" in Chapter 2, "Indexing".

CREATE_SECTION_GROUP

ADD_FIELD_SECTION

ADD_SPECIAL_SECTION

REMOVE_SECTION

DROP_SECTION_GROUP

CREATE_INDEX_SET

Creates an index set for CTXCAT index types. You 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. You 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 ORA:CONTAINS function. ORA:CONTAINS is a function you use within an XPATH query expression with existsNode.

See Also: *Oracle9i Application Developer's Guide - XML*

Syntax

```
CTX_DDL.CREATE_POLICY(
         policy_name     IN VARCHAR2 DEFAULT NULL,
         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.

filter
Specify the filter preference to use.

Note: In this release, this parameter is not used.

section_group
Specify the section group to use. You can specify only NULL_SECTION_GROUP. Only special (sentence and paragraph) section are supported.

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 you can issue 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;
```

You can update your policy by doing:

```
exec ctx_ddl.update_policy(policy_name => 'my_policy', lexer => 'my_new_lex');
```

You can drop your policy by doing:

```
exec ctx_ddl.drop_policy(policy_name => 'my_policy');
```

CREATE_PREFERENCE

Creates a preference in the Text data dictionary. You specify preferences in the parameter string of CREATE INDEX or ALTER INDEX.

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, "Indexing".

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 Y for the INDEX_TEXT attribute, then specifying N 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" in Chapter 2, "Indexing".

Creating Master/Detail Relationship

You can use CTX_DDL.CREATE_PREFERENCE to create a preference with DETAIL_DATASTORE. You 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" in Chapter 2, "Indexing".

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', '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" in Chapter 2, "Indexing".

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;
```

Related Topics

SET_ATTRIBUTE

DROP_PREFERENCE

CREATE INDEX in Chapter 1, "SQL Statements and Operators".

ALTER INDEX in Chapter 1, "SQL Statements and Operators".

Chapter 2, "Indexing"

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, or ADD_SPECIAL_SECTION.

When you index, you name the section group in the parameter string of CREATE INDEX or ALTER INDEX.

After indexing, you can 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 characters. 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 operator in Chapter 3, "CONTAINS Query Operators".

"Section Group Types" in Chapter 2, "Indexing".

ADD_ZONE_SECTION

ADD_FIELD_SECTION

ADD_SPECIAL_SECTION

REMOVE_SECTION

DROP_SECTION_GROUP

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, you text table must contain a language column.

You can add either stopwords, stopclasses, or stopthemes to a stoplist using ADD_STOPWORD, ADD_STOPCLASS, or ADD_STOPTHEME.

You can 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, your table must have a language column.

Example

Single Language Stoplist

The following code creates a stoplist called `mystop`:

```
begin
ctx_ddl.create_stoplist('mystop', 'BASIC_STOPLIST');
end;
```

Multi-Language Stoplist

The following code 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

ADD_STOPCLASS

ADD_STOPTHEME

DROP_STOPLIST

CREATE INDEX in Chapter 1, "SQL Statements and Operators".

ALTER INDEX in Chapter 1, "SQL Statements and Operators".

Appendix D, "Supplied Stoplists"

DROP_INDEX_SET

Drops an index set.

Syntax

```
CTX_DDL.DROP_INDEX_SET(set_name in varchar2);
```

set_name
Specify the name of the index set to drop.

DROP_POLICY

Drops a policy create with 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 code drops the preference my_lexer.

```
begin
ctx_ddl.drop_preference('my_lexer');
end;
```

Related Topics

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.

Examples

The following code drops the section group htmgroup and all its sections:

```
begin
ctx_ddl.drop_section_group('htmgroup');
end;
```

Related Topics

CREATE_SECTION_GROUP

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 code drops the stoplist `mystop`:

```
begin
ctx_ddl.drop_stoplist('mystop');
end;
```

Related Topics

CREATE_STOPLIST

OPTIMIZE_INDEX

Use this procedure to optimize the index. You optimize your index after you synchronize it. Optimizing the index removes old data and minimizes index fragmentation. Optimizing the index can improve query response time.

You can optimize in fast, full, or token mode. In token mode, you specify a specific token to be optimized. You can use token 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 for that token.

> **Note:** Optimizing an index can result in better response time only if you insert, delete, or update documents in your base table after your initial indexing operation.

Using this procedure to optimize your index is recommended over using the ALTER INDEX statement.

Limitations

The CTX_DDL.OPTIMIZE_INDEX procedure optimizes at most 16,000 document ids. To continue optimizing more document ids, re-run this procedure.

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,
    parallel_degree IN VARCHAR2);
);
```

idx_name
Specify the name of the index. If you do not specify an index name, Oracle chooses a single index to optimize.

optlevel
Specify optimization level as a string. You can specify one of the following methods for optimization:

Value	Description
FAST or CTX_DDL.OPTLEVEL_FAST	This method compacts fragmented rows. However, old data is not removed.
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.
TOKEN	This method lets you specify a specific token to be optimized. Oracle does a FULL optimization on the token you specify with *token*. Use this method to optimize those tokens that are searched frequently. Token optimization is not supported for CTXRULE indexes.

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
Specify the name of the index partition to optimize.

parallel_degree
Specify the parallel degree as a number for parallel optimization. The actual parallel degree depends on your resources.

Examples

The following two examples optimize the index for 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;
```

Related Topics

ALTER INDEX in Chapter 1, "SQL Statements and Operators".

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_SECTION

The REMOVE_SECTION procedure removes the specified section from the specified section group. You can specify the section by name or by id. You can 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.

Examples

The following code drops a section called Title from the htmgroup:

```
begin
ctx_ddl.remove_section('htmgroup', 'Title');
end;
```

Related Topics

ADD_FIELD_SECTION

ADD_SPECIAL_SECTION

ADD_ZONE_SECTION

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 code removes the stopclass `NUMBERS` from the stoplist `mystop`.

```
begin
ctx_ddl.remove_stopclass('mystop', 'NUMBERS');
end;
```

Related Topics

ADD_STOPCLASS

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 code removes the stoptheme *banking* from the stoplist `mystop`:

```
begin
ctx_ddl.remove_stoptheme('mystop', 'banking');
end;
```

Related Topics

ADD_STOPTHEME

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-supported language. You can also remove ALL stopwords.

Example

The following code removes a stopword *because* from the stoplist `mystop`:

```
begin
    ctx_ddl.remove_stopword('mystop','because');
end;
```

Related Topics

ADD_STOPWORD

SET_ATTRIBUTE

Sets a preference attribute. You 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. You can 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" in Chapter 2, "Indexing".
>
> For more examples of using SET_ATTRIBUTE, see CREATE_PREFERENCE.

SYNC_INDEX

Synchronizes the index to process inserts, updates, and deletes to the base table.

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);
```

idx_name
Specify the name of the index.

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 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

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

part_name
Specify the name of the index partition to synchronize.

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.

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;
```

Related Topics

ALTER INDEX in Chapter 1, "SQL Statements and Operators".

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

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 DEFAULT NULL,
         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.

> **Note:** You can use this package only when your index type is CONTEXT. This package does not support the CTXCAT index type.

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
SET_KEY_TYPE	Sets CTX_DOC procedures to accept rowid or primary key document identifiers.
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.

Syntax 1:In-memory Result Storage

```
CTX_DOC.FILTER(
         index_name  IN VARCHAR2,
         textkey     IN VARCHAR2,
         restab      IN OUT NOCOPY CLOB,
plaintext   IN BOOLEAN  DEFAULT FALSE);
```

Syntax 2: Result Table Storage

```
CTX_DOC.FILTER(
         index_name  IN VARCHAR2,
         textkey     IN VARCHAR2,
         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 one of the following:

- 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

You 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 result table must exist before you make this call.

> **See Also:** "Filter Table" in Appendix A, "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, 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 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 INSO 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, issue the following statement:

```
begin
ctx_doc.filter('newsindex', '20', 'filtertab', '0', TRUE);
end;
```

GIST

Use the CTX_DOC.GIST procedure to generate a gist and theme summaries for a document. You can generate paragraph-level or sentence-level gists or theme summaries.

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 NULL,
    maxPercent    IN NUMBER DEFAULT NULL,
    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 NULL,
    maxPercent    IN NUMBER DEFAULT NULL,
    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 be one of the following:

- 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

You toggle between primary key and rowid identification using CTX_DOC.SET_KEY_TYPE.

restab
You can 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 the table.

> **See Also:** "Gist Table" in Appendix A, "Result Tables" for more information about the structure of the gist result table, see

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 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 and you do not specify a value for pov, this procedure returns the generic gist plus up to fifty 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 pov, 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 0.

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 0.

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.

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. Yo can 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 in either an in-memory PL/SQL table or a result table.

Syntax 1:In-Memory Result Storage

```
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);
```

Syntax 2:Result Table Storage

```
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);
```

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 one of the following:

- 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

You 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:** see "Highlight Table" in Appendix A, "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 INSO 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, you 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.

IFILTER

Use this procedure when you need to filter binary data to text.

This procedure takes binary data (BLOB IN), filters the data through with the Inso filter, and writes the text version to a CLOB. CTX_DOC.IFILTER employs the safe callout, so can be called from a user datastore procedure, and it does not require an index to use, as CTX_DOC.FILTER does.

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.

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.

You can use one of the pre-defined tagsets 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.

Syntax 1: In-Memory Result Storage

```
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);
```

Syntax 2: Result Table Storage

```
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);
```

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 one of the following:

- 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

You 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, "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 INSO filter or indexing HTML documents.

tagset
Specify one of the following pre-defined tagsets. 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	

Tagset	Tag	Tag Value
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 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.

Examples

In-Memory Markup

The following code generates a marked-up document and stores it in memory. The code passes a NULL CLOB locator to MARKUP and then de-allocates the returned CLOB locator after using it.

```
set serveroutput on

declare
mklob clob;
amt number := 40;
line varchar2(80);

begin
 ctx_doc.markup('myindex','1','dog & 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;
```

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

To create HTML highlight markup for the words *dog* or *cat* for document 23, issue 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, issue 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;
```

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.

Examples

```
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.

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 you specify for any CTX_DOC procedure is interpreted as a ROWID.

Example

To set CTX_DOC procedures to accept primary key document identifiers, do the following:

```
begin
ctx_doc.set_key_type('PRIMARY_KEY');
end
```

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 you specify.

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 one of the following:

- 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

You 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.

To store results in a table, specify the name of the table.

> **See Also:** "Theme Table" in Appendix A, "Result Tables" for more information about the structure of the theme result table.

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 number of themes to retrieve. For example, if you specify 10, the top 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 top 50 themes show conceptual hierarchy.

Examples

In-Memory Themes

The following example generates the top 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, numthemes=>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 the top 20 themes where each element in the list is a single theme, issue a statement like the following:

```
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, issue a statement like the following:

```
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 which 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 one of the following:

- 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

You 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 which 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 following columns, with names and data types as specified.

Table 8–1

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`.

Examples

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	Add an event to the index log.
END_LOG	Halts logging of index and document services requests.
LOGFILENAME	Returns the name of the current log file.
REMOVE_EVENT	Remove an event from the index log.
START_LOG	Starts logging index and document service requests.

ADD_EVENT

Use this procedure to add an event to the index log for more detailed log output. Currently the only event you can add is the 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.

Syntax

```
CTX_OUTPUT.ADD_EVENT(event in varchar2);
```

event
Specify the type of index event to log. Currently the only event you can add is the CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID which logs the rowid of each row after it is indexed.

Example

```
begin
    CTX_OUTPUT.ADD_EVENT(CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID);
end;
```

END_LOG

Halt logging index and document service requests

Syntax

```
CTX_OUTPUT.END_LOG;
```

Example

```
begin
CTX_OUTPUT.END_LOG;
end;
```

LOGFILENAME

Returns the filename for the current log. This procedure looks for the log file in the directory specified by the LOG_DIRECTORY system parameter.

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 varchar2);
```

event
Specify the type of index event to remove from the log. Currently the only event you can add and remove is the CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID.

Example

```
begin
    CTX_OUTPUT.REMOVE_EVENT(CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID);
end;
```

START_LOG

Begin logging index and document service requests.

Syntax

```
CTX_OUTPUT.START_LOG(logfile in varchar2);
```

logfile
Specify the name of the log file. The log is stored in the directory specified by the system parameter LOG_DIRECTORY.

Example

```
begin
CTX_OUTPUT.START_LOG('mylog1');
end;
```

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.
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. You specify a seed word and BROWSE_WORDS returns the words around it in the index, and a rough count of the number of documents that contain each word.

This feature is useful for refining queries. You can identify the following:

- 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, it defaults to 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

```
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;
```

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.

part_name
Specify the name of the index partition to query.

Notes

If the query contains structured criteria, you should use SELECT COUNT(*).

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 issue 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 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.

Limitation

You cannot use EXPLAIN with remote queries.

Syntax

```
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
);
```

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:** For more information about the structure of the explain table, see "EXPLAIN Table" in Appendix A, "Result Tables".

sharelevel
Specify whether explain_table is shared by multiple `EXPLAIN` calls. Specify 0 for exclusive use and 1 for shared use. This parameter defaults to 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. This parameter defaults to `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);
```

Executing 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
```

Related Topics

Chapter 3, "CONTAINS Query Operators"

Appendix H, "Stopword Transformations"

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 is only supported in English and French.

Syntax

```
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
           );
```

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:** For more information about the structure of the explain table, see "HFEEDBACK Table" in Appendix A, "Result Tables".

sharelevel
Specify whether feedback_table is shared by multiple HFEEDBACK calls. Specify 0 for exclusive use and 1 for shared use. This parameter defaults to 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. This parameter defaults to 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 ctx_feedback_type,
  rt_feedback ctx_feedback_type,
  nt_feedback 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: For more information about the structure of the hfeedback table, see "HFEEDBACK Table" in Appendix A, "Result Tables".

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 example above, 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 above.

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.

Examples

```
begin
ctx_query.remove_sqe('disasters');
end;
```

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

```
CTX_QUERY.STORE_SQE(query_name      IN VARCHAR2,
                    text_query      IN VARCHAR2);
```

query_name
Specify the name of the stored query expression to be created. If you are CTXSYS, you can specify this as user.name.

text_query
Specify the query expression to be associated with query_name.

Examples

```
begin
ctx_query.store_sqe('disasters', 'hurricanes | earthquakes');
end;
```

11

CTX_REPORT

This chapter describes how to use the CTX_REPORT package to create various index reports. This chapter contains the following topics:

- Procedures in CTX_REPORT
- Using the Function Versions

Procedures in CTX_REPORT

The CTX_REPORT package contains the following procedures:

Name	Description
DESCRIBE_INDEX	Create a report describing the index.
DESCRIBE_POLICY	Create 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.
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 variants. 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 meta-data, 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
);

function CTX_REPORT.DESCRIBE_INDEX(
  index_name IN VARCHAR2
) 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.

DESCRIBE_POLICY

Creates a report describing the policy. This includes the settings of the policy meta-data, 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
);

function CTX_REPORT.DESCRIBE_POLICY(
  policy_name IN VARCHAR2
) return CLOB;
```

policy_name
Specify the name of the policy 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.

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.

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
);

function CTX_REPORT.INDEX_SIZE(
  index_name  IN VARCHAR2,
  part_name   IN VARCHAR2 DEFAULT NULL
) 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.

INDEX_STATS

Creates a report showing various calculated statistics about the text index.

This procedure will fully scan the text index tables, so it may take a long time to run for large indexes.

INDEX_STATS will create and use a session-duration temporary table, which will be created in CTXSYS temp tablespace.

```
procedure 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
);
```

index_name
Specify the name of the index to describe. You can specify a CONTEXT, CTXCAT, CTXRULE, or CTXXPATH 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.

Example

The following is sample output 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 per 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 per 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 per 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 %
```

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
);

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'
) 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. 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.

resolve_docids
Specify TRUE to resolve docids to rowids.

To facilitate inline invocation, the boolean arguments are varchar2 in the function variant. You can pass in 'Y', 'N', 'YES', 'NO', 'T', 'F', 'TRUE', or 'FALSE'

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_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
ATTR TEXT	Text that occurs in attribute.	4
ATTR SEC	Attribute section.	5
PREFIX	Prefix token.	6
PATH SEC	Path section.	7

Input	Meaning	Type Returned
PATH ATTR	Path attribute section.	8
STEM	Stem form token.	9
FIELD <name> TEXT	Text token in field section <name>	16-79
FILED <name> PREFIX	Prefix token in field section <name>	616-916
FIELD <name> STEM	Stem token in field section <name>	916-979

For FIELD types, the index meta-data 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.

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.

Name	Description
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.
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, "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 thesaurus name.

phrase
Specify 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 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 new scope note.

operand
Specify 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 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, "Result Tables" for more information about EXP_TAB.

phrase
Specify 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 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, issue 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, "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, "Result Tables" for more information about EXP_TAB.

phrase
Specify 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, issue 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, "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, "Result Tables" for more information about EXP_TAB.

phrase
Specify 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 instance in the form:

```
{bt1}|{bt2}|{bt3} ...
```

Example

To look up the broader terms instance for *cat* up to two levels, issue 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, "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, "Result Tables" for more information about EXP_TAB.

phrase
Specify 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, 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 2 broader terms partitive for *cat*, issue 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, "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.

Examples

Creating Entries for Phrases

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.

> **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 related phrase. The relationship is interpreted from phrase to relphrase.
RT	Add associative relation.	Specify 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 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 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 phrase to drop. If phrase is a disambiguated homograph, you must include the qualifier. When phrase does not exist in tname, this procedure raises and 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 thesaurus name.

phrase
Specify the filing phrase.

rel
Specify 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)

relation	meaning	relphrase
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.

Examples

```
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-separated 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, "Result Tables" for more information about EXP_TAB.

phrase

Specify 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, issue 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, "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, "Result Tables" for more information about EXP_TAB.

phrase
Specify 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 generic in the form:

```
{nt1}|{nt2}|{nt3} ...
```

Example

To look up the narrower terms generic for *cat* down to two levels, issue 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, "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, "Result Tables" for more information about EXP_TAB.

phrase
Specify 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 instance in the form:

```
{nt1}|{nt2}|{nt3} ...
```

Example

To look up the narrower terms instance for *cat* down to two levels, issue 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, "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, "Result Tables" for more information about EXP_TAB.

phrase
Specify 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 partitive in the form:

```
{nt1}|{nt2}|{nt3} ...
```

Example

To look up the narrower terms partitive for *cat* down to two levels, issue 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, "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, "Result Tables" for more information about EXP_TAB.

phrase
Specify 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, "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, "Result Tables" for more information about EXP_TAB.

phrase
Specify phrase to lookup in thesaurus.

tname
Specify 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, "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 phrase to lookup in thesaurus.

tname
Specify 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, "Result Tables" for more information about EXP_TAB.

phrase
Specify phrase to lookup in thesaurus.

tname
Specify 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, issue 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, "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

Since 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, "Result Tables" for more information about EXP_TAB.

tname
Specify 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 thesuarus, 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, "Result Tables" for more information about EXP_TAB.

phrase

Specify phrase to lookup in thesaurus.

lang
Specify the foreign language. Specify `'ALL'` for all translations of `phrase`.

tname
Specify 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*, you can issue 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, "CONTAINS Query Operators"

TRSYN

For a given mono-lingual thesuarus, 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, "Result Tables" for more information about EXP_TAB.

phrase

Specify phrase to lookup in thesaurus.

lang
Specify the foreign language. Specify `'ALL'` for all translations of *phrase*.

tname
Specify 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*, you can issue 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, "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, "Result Tables" for more information about EXP_TAB.

phrase
Specify phrase to lookup in thesaurus.

tname
Specify 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, "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 for using the CTX_ULEXER PL/SQL package to use with the user-lexer.

CTX_ULEXER declares the following type:

Name	Description
WILDCARD_TAB	Index-by table type 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.

14

Executables

This chapter discusses the executables shipped with Oracle Text. The following topics are discussed:

- Thesaurus Loader (ctxload)
- Knowledge Base Extension Compiler (ctxkbtc)

Thesaurus Loader (ctxload)

Use `ctxload` to do the following with a thesaurus:

- import a thesaurus file into the Oracle Text thesaurus tables.
- export a loaded thesaurus to a user-specified operating-system file.

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" in Appendix C, "Loading Examples".

Text Loading

The `ctxload` program no longer supports the loading of text columns. To load files to a text column in batch, Oracle recommends that you use SQL*Loader.

> **See Also:** "SQL*Loader Example" in Appendix C, "Loading Examples"

ctxload Syntax

```
ctxload -user username[/password][@sqlnet_address]
        -name object_name
        -file file_name

       [-thes]
       [-thescase y|n]
       [-thesdump]
       [-log file_name]
       [-trace]
       [-pk]
       [-export]
       [-update]
```

Mandatory Arguments

-user
Specify the username and password of the user running ctxload.

The username and password can be followed immediately by *@sqlnet_address* to permit logon 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, you do not have to specify a value for *sqlnet_address* to connect to the database.

-name object_name
When you use ctxload to export/import a thesaurus, use object_name to specify the name of the thesaurus to be exported/imported.

You 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, ctxload returns an error and does not overwrite the existing thesaurus.

When you use ctxload to update/export a text field, use object_name to specify the index associated with the text column.

-file file_name
When ctxload is used to import a thesaurus, use file_name to specify the name of the import file which 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, ctxload *overwrites* the existing file.

Optional Arguments

-thes
Import a thesaurus. Specify the source file with the -file argument. You 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 with only 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 *Oracle9i Database Administrator's Guide*

-pk
Specify the primary key value of the row to be updated or exported.

When the primary key is compound, you must enclose the values within double quotes and separate the keys with a comma.

-export
Exports the contents of a `CLOB` or `BLOB` column in a database table into the operating system file specified by -file. `ctxload` exports the `CLOB` or `BLOB` column in the row specified by -pk.

When you use the -export, you must specify a primary key with -pk.

-update
Updates the contents of a `CLOB` or `BLOB` column in a database table with the contents of the operating system file specified by -file. ctxload updates the `CLOB` or BLOB column in for the row specified by -pk.

When you use -update, you must specify a primary key with -pk.

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, "Loading Examples".

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 Oracle Text uses to perform theme analysis, such as theme indexing, processing ABOUT queries, and document theme extraction with the CTX_DOC package. A knowledge base is supplied for English and French.

With the ctxkbtc compiler, you can do the following:

- 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.

> **See Also:** For more information about the knowledge base packaged with Oracle Text, see Appendix I, "English Knowledge Base Category Hierarchy".
>
> For more information about the ABOUT operator, see ABOUT operator in Chapter 3, "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 username 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 name(s) 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. You can still 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, since 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, Oracle recommends that you recompile your extended knowledge base in the new environment for theme indexing and related features to work correctly.
- Knowledge base extension cannot be performed when theme indexing is being performed. In addition, any SQL sessions that are using Oracle Text functions must be exited and reopened to make use of the extended knowledge base.
- There can be only one user extension per installation. Since a user extension affects all users at the installation, only administrators or terminology managers should extend the knowledge base.

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 will tolerate certain violations of the above rules. For example, if a term has multiple BTs, it ignores all but the last one it encounters.
>
> Similarly, BTs between existing knowledge base categories will only result in a warning message.
>
> Such violations are not recommended since they might produce undesired results.

Extending the Knowledge Base

You can 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 more information about the knowledge base, see Appendix I, "English Knowledge Base Category Hierarchy"

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.

You 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, you 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

You can 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.

You extend theme functionality by adding a user-defined knowledge base. For example, you can create a Spanish knowledge base from a Spanish thesuarus.

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 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 multi-byte character sets. If the database is a multi-byte 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 per 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 (most preferred first). A user 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:

Description of Parameter	Limit
Number of RTs (from + to) per term	32
Number of terms per a single hierarchy (i.e., 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

A

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

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.
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.

Table A–1

Column Name	Datatype	Description
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

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

The following table list the possible values for the OPTIONS column of the explain table.

Table A–3

Options Value	Description
($)	Stem
(?)	Fuzzy
(!)	Soundex
(T)	Order for ordered Near.
(F)	Order for unordered Near.
(n)	A number associated with the max_span parameter for the Near operator.

HFEEDBACK Table

Table A–4 describes the table to which CTX_QUERY.HFEEDBACK writes its results.

Table A–4

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

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

The following table list the values for the OPTIONS column of the feedback table.

Table A–6

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 Members and Methods	Type	Description
text	member	Feedback term.
cardinality	member	(reserved for future use.)
score	member	(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

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

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

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

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

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

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. You can 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. The following table describes the fields in EXP_REC:

EXP_REC Field	Description
xrel	The xrel field contains the relation of the term to the input term (e.g. 'SYN', 'PT', 'RT', etc.). 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

Supported Document Formats

This appendix contains a list of the document formats supported by the Inso filtering technology. The following topics are covered in this appendix:

- About Document Filtering Technology
- Supported Document Formats
- Unsupported Formats

About Document Filtering Technology

Oracle Text uses document filtering technology licensed from Stellent Chicago, Inc. This filtering technology enables you to index most document formats. This technology also enables you to convert documents to HTML for document presentation, with the CTX_DOC package.

> **See Also:** For a list of supported formats, see "Supported Document Formats" in this Appendix.

To use Inso filtering for indexing and DML processing, you must specify the INSO_FILTER object in your filter preference.

To use Inso filtering technology for converting documents to HTML with the CTX_DOC package, you need not use the INSO_FILTER indexing preference, but you must still set up your environment to use this filtering technology as described in this appendix.

To convert documents to HTML format, Inso filtering technology relies on shared libraries and data files licensed from Stellent Chicago, Inc.

The following sections discuss the supported platforms and how to enable Inso filtering on the different platforms.

Supported Platforms

Supported Platforms

Inso filter technology is supported on the following platforms:

- Sun Solaris on SPARC 32-bit and 64-bit (2.5.1 - 2.6,7-8)
- IBM AIX 32-bit and 64-bit (4.2 - 4.3)
- HP-UX 32-bit and 64-bit (10.2 - 11.0)
- DEC UNIX for Alpha / Tru64 UNIX (4.0)
- SGI IRIX 32-bit and 64-bit (6.3)
- Microsoft Windows
  - Intel x86 WinNT (4.0 and above)
  - Intelx86 Win95, Win98 SE, Win2000, and Windows ME
- Red Hat Linux for Intel x86 (5.2 - 7.0)

Environment Variables

All environment variables related to Inso filtering must be made visible to Oracle Text.

Requirements for UNIX Platforms

The following requirements apply to Solaris, IBM AIX, HP/UX, Digital UNIX, SGI, and Linux platforms:

- Ensure the *.flt files have execute permission granted to the operating system user running the Oracle database and ctxsrv server.
- Set the `$PATH` variable to include the location of the *.flt files, in particular to the location of the file `isunx2.flt`, and to `$ORACLE_HOME/ctx/lib` which is the location of the shared libraries for Inso filtering
- Set the `$HOME` environment variable to allow Inso technology to write files to a sub-directory (.oit) in `$HOME` directory.
- Access to a running X-Windows server is required to perform vector graphics image conversion.

Filtering Vector Graphic Formats

Follow these steps to filter vector graphic formats on UNIX platforms:

- Start an X server to filter vector graphic formats. If no X server exists (system detects no X libraries, such as Xm, Xt, and X11), vector graphic filtering is not performed. Vector graphic formats include CAD drawings and presentation formats such as Power Point 97. Bitmap formats include GIF, JPEG, and TIF formats as well as bitmap formats.
- Because the system depends on X libraries to perform vector graphic conversion, ensure that the system-specific library path environment variable for the X libraries is set correctly.
- Set the `$DISPLAY` environment variable. For example, setting `DISPLAY=:0.0` tells the system to use the X server on the console.

OLE2 Object Support

There are platform dependent limits on what Inso filter technology can do with OLE2 objects. On all platforms when a metafile snapshot is available, Inso technology will use it to convert the object.

When a metafile snapshot is not available on UNIX platforms, Inso technology cannot convert the OLE2 object.

However, when a metafile snapshot is not available on the NT platform, the original application is used (if available) to convert the OLE2 object.

Supported Document Formats

The following table lists all of 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. This filtering technology is based on Outside In HTML Export and Outside In Content Access technology licensed from Stellent Chicago, Inc.

Note: This list does *not* represent the complete list of formats that Oracle is able to process. The external filter framework enables Oracle to process *any* document format, provided an external filter exists which can filter all the formats to plain text.

Word Processing - Generic

Format	Version
ASCII Text (7 &8 bit versions)	All versions
ANSI Text (7 & 8 bit)	All versions
Unicode Text	All versions
HTML	Versions through 3.0 (some limitations)
IBM Revisable Form Text	All versions
IBM FFT	All versions
Microsoft Rich Text Format (RTF)	All versions

Word Processing - DOS

Format	Version
DEC WPS Plus (WPL)	Versions through 4.1
DEC WPS Plus (DX)	Versions through 4.0
DisplayWrite 2 & 3 (TXT)	All versions
DisplayWrite 4 & 5	Versions through Release 2.0
Enable	Versions 3.0, 4.0 and 4.5

Format	Version
First Choice	Versions through 3.0
Framework	Version 3.0
IBM Writing Assistant	Version 1.01
Lotus Manuscript	Versions through 2.0
MASS11	Versions through 8.0
Microsoft Word	Versions through 6.0
Microsoft Works	Versions through 2.0
MultiMate	Versions through 4.0
Navy DIF	All versions
Nota Bene	Version 3.0
Office Writer	Version 4.0 to 6.0
PC-File Letter	Versions through 5.0
PC-File+ Letter	Versions through 3.0
PFS:Write	Versions A, B, and C
Professional Write	Versions through 2.1
Q&A	Version 2.0
Samna Word	Versions through Samna Word IV+
SmartWare II	Version 1.02
Sprint	Versions through 1.0
Total Word	Version 1.2
Volkswriter 3 & 4	Versions through 1.0
Wang PC (IWP)	Versions through 2.6
WordMARC	Versions through Composer Plus
WordPerfect	Versions through 6.1
WordStar	Versions through 7.0
WordStar 2000	Versions through 3.0
XyWrite	Versions through III Plus

Word Processing - International

Format	Version
JustSystems Ichitaro	Version 5.0, 6.0, 8.0, 9.0, and 10.0

Word Processing - Windows

Format	Version
AMI / AMI Professional	Versions through 3.1
Corel WordPerfect for Windows	Versions through 2002
JustWrite	Versions through 3.0
Legacy	Versions through 1.1
Lotus WordPro (NT on Intel only)	SmartSuite 96, 97, Millennium and Millennium 9.6
Lotus WordPro (all supported platforms except NT on Intel; Text only)	SmartSuite 97, Millennium, and Millennium 9.6
Microsoft Windows Works	Versions through 4.0
Microsoft Windows Write	Versions through 3.0
Microsoft Word 97	Word 97
Microsoft Word 2000	Word 2000
Microsoft Word 2002 (Office XP)	Word 2002
Microsoft Word for Windows	Versions through 7.0
Microsoft WordPad	All versions
Novell Perfect Works	Version 2.0
Novell WordPerfect for Windows	Versions through 7.0
Professional Write Plus	Version 1.0
Q&A Write for Windows	Version 3.0
Star Office Writer for Windows (Text only)	Version 5.2
WordStar for Windows	Version 1.0

Word Processing - Macintosh

Format	Version
Microsoft Word	Versions 4.0 through 6.0
Microsoft Word 98	Word 98
WordPerfect	Versions 1.02 through 3.0
Microsoft Works	Versions through 2.0
MacWrite II	Version 1.1

Word Processing - Unix

Format	Version
Star Office Writer for Windows	Version 5.2

Desktop Publishing

Format	Version
Adobe FrameMaker	Version 6.0

Spreadsheets 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 for SmartSuite	SmartSuite 97, Millennium, and Millennium 9.6
Lotus 1-2-3 Charts (DOS & Windows)	Versions through Millennium 9.6
Lotus 1-2-3 (OS/2)	Versions through 2.0

Format	Version
Lotus 1-2-3 Charts (OS/2)	Versions through 2.0
Lotus Symphony	Versions 1.0,1.1 and 2.0
Microsoft Excel 97	Excel 97
Microsoft Excel 2000	Excel 2000
Microsoft Excel 2002 (Office XP)	Excel 2002
Microsoft Excel Windows	Versions 2.2 through 7.0
Microsoft Excel Macintosh	Versions 3.0 - 4.0 and 98
Microsoft Excel Charts	Versions 2.x - 7.0
Microsoft Multiplan	Version 4.0
Microsoft Windows Works	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
QuattroPro for DOS	Versions through 5.0
QuttroPro for Windows	Versions through 2002
PFS:Professional Plan	Version 1.0
SuperCalc 5	Version 4.0
SmartWare II	Version 1.02
VP Planner 3D	Version 1.0

Databases Formats

Format	Version
Access	Versions through 2.0
dBASE	Versions through 5.0
DataEase	Version 4.x
dBXL	Version 1.3

Format	Version
Enable	Versions 3.0, 4.0 and 4.5
First Choice	Versions through 3.0
FoxBase	Version 2.1
Framework	Version 3.0
Microsoft Windows Works	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

Display Formats

Format	Version
PDF - Portable Document Format	Acrobat Versions 2.1, 3.0, 4.0, and 5.0 including Japanese PDF.

Presentation Formats

Format	Version
Corel Presentations	Versions 8.0, 9.0 and 2002
Novell Presentations	Versions 3.0 and 7.0
Harvard Graphics for DOS	Versions 2.x & 3.x
Harvard Graphics	Windows versions

Format	Version
Freelance 96	Freelance 96
Freelance for Windows	SmartSuite 97, Millennium, and Millennium 9.6
Freelance for Windows	Version 1.0 and 2.0
Freelance for OS/2	Versions through 2.0
Microsoft PowerPoint for Windows	Versions through 7.0
Microsoft PowerPoint 97	PowerPoint 97
Microsoft PowerPoint 2000	PowerPoint 2000
Microsoft PowerPoint 2002 (Office XP)	PowerPoint 2002
Microsoft PowerPoint for Macintosh	Version 4.0 and 98

Standard Graphic Formats

The following table lists the graphic formats that the INSO 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.

Note: The INSO filter cannot extract textual information from graphics.

Format	Version
Binary Group 3 Fax	All versions
BMP (including RLE, ICO, CUR & OS/2 DIB)	Windows
CALS Raster	Type 1 and II
CDR (if TIFF image is embedded in it)	Corel Draw version 2.0 - 9.0
CGM - Computer Graphics Metafile	ANSI, CALS, NIST, Version 3.0
DCX (multi-page PCX)	Microsoft Fax
DRW - Micrografx Designer	Version 3.1
DRW - Micrografx Draw	Version 4.0
DXF (Binary and ASCII) AutoCAD Drawing Interchange Format	Versions through 14

Format	Version
EMF	Windows Enhanced Metafile
EPS - Encapsulated PostScript	If TIFF image is embedded in it
FPX - Kodak Flash Pix	No specific version
GIF - Graphics Interchange Format	Compuserve
GP4 - Group 4 CALS format	Types I and II
HPGL - Hewlett Packard Graphics Language	Version 2.0
IMG - GEM Paint	No specific version
JFIF (JPEG not in TIFF)	All versions
JPEG	All versions
Novell Perfect Works (Draw)	Novell version 2.0
PBM - Portable Bitmap	No specific version
PCD - Kodak Photo CD	Version 1.0
PCX Bitmap	PC Paintbrush
PGM - Portable Graymap	No specific version
PIC	Lotus 1-2-3 Picture File Format - No Specific Version
PICT1 & PICT2 (Raster)	Macintosh Standard
PNG - Portable Network Graphics Internet Format	Version 1.0
PNTG	MacPaint
PPM - Portable Pixmap	No specific version
Progressive JPEG	No Specific version
PSP - Paintshop Pro (NT on Intel only)	Versions 5.0 and 5.0.1
SDW	Ami Draw
Snapshot (Lotus)	All versions
SRS - Sun Raster File Format	No specific version
Targa	Truevision
TIFF	Versions through 6
TIFF CCITT Group 3 & 4	Fax Systems

Format	Version
VISO	Visio 4 (Page Preview only), 5, 2000, 2002
WBMP	No Specific version
WMF	Windows Metafile
WordPerfect Graphics [WPG and WPG2]	Versions through 2.0
XBM - X-Windows Bitmap	x10 compatible
XPM - X-Windows Pixmap	x10 compatible
XWD - X-Windows Dump	x10 compatible

Other

Format	Version
Executable (EXE, DLL)	No specific version
Executable for Windows NT	No specific version
Microsoft Project (Text only)	Project 98
MSG (Text only)	Microsoft Outlook mail format
vCard Electronic Business Card	Versit version 2.1
WML	Compatible with version 5.2

Unsupported Formats

Password protected documents and documents with password protected content are not supported by the Inso filter.

C

Loading Examples

This appendix provides examples of how to load text into a text column. It also describes the structure of `ctxload` import files:

- 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
- issue the SQL*Loader command that reads control file and loads data into table

> **See Also:** For a complete discussion on using SQL*Loader, see Oracle9i Database Utilities

Creating the Table

This example loads to 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, and / 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.

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 (i.e. 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 characters, 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, etc.) 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 characters, 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

Supplied Stoplists

This appendix describes the default stoplists for all the different languages supported and list the stopwords in each. The following stoplists are described:

- English Default Stoplist
- Chinese Stoplist (Traditional)
- Chinese Stoplist (Simplified)
- Danish (dk) Default Stoplist
- Dutch (nl) Default Stoplist
- Finnish (sf) Default Stoplist
- French (f) Default Stoplist
- German (d) Default Stoplist
- Italian (i) Default Stoplist
- Portuguese (pt) Default Stoplist
- Spanish (e) Default Stoplist
- Swedish (s) Default Stoplist

English Default Stoplist

The following English words are defined as stop words:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	be	had	it	only	she	was
about	because	has	its	of	some	we
after	been	have	last	on	such	were
all	but	he	more	one	than	when
also	by	her	most	or	that	which
an	can	his	mr	other	the	who
any	co	if	mrs	out	their	will
and	corp	in	ms	over	there	with
are	could	inc	mz	s	they	would
as	for	into	no	so	this	up
at	from	is	not	says	to	

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
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:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
aina	hyvin	kesken	me	nyt	takia	yhdessä
alla	hoikein	kukka	mikä	oikea	tässä	ylös
ansiosta	ilman	kyllä	miksi	oikealla	te	
ei	ja	kylliksi	milloin	paljon	ulkopuolella	
enemmän	jälkeen	tarpeeksi	milloinkan	siellä	vähän	
ennen	jos	lähellä	koskaan	sinä	vahemmän	
etessa	kanssa	läpi	minä	ssa	vasen	
haikki	kaukana	liian	missä	sta	vasenmalla	
hän	kenties	lla	miten	suoraan	vastan	
he	ehkä	luona	kuinkan	tai	vielä	
hitaasti	keskellä	lla	nopeasti	takana	vieressä	

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
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	zwischens
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:

Stop word	Stop word	Stop word	Stop word
ab	efter	ja	sin
aldrig	efterät	jag	skall
all	eftersom	långsamt	som
alla	ej	långt	till
alltid	eller	lite	tillräckligt
än	emot	man	tillsammans
ännu	en	med	trots att
ånyo	ett	medan	under
är	fastän	mellan	uppe
att	för	mer	ut
av	fort	mera	utan
avser	framför	mindre	utom
avses	från	mot	vad
bakom	genom	myckett	väl
bra	gott	när	var
bredvid	hamske	nära	varför
dä	han	nej	vart
där	här	nere	varthän
de	hellre	ni	vem
dem	hon	nu	vems
den	hos	och	vi
denna	hur	oksa	vid
deras	i	om	vilken
dess	in	över	
det	ingen	på	
detta	innan	så	
du	inte	sådan	

E

Alternate Spelling Conventions

This appendix describes the alternate spelling conventions that Oracle Text uses in the German, Danish, and Swedish languages. This chapter also describe how to enable alternate spelling.

The following topics are covered:

Overview

This chapter lists the alternate spelling conventions Oracle Text uses for German, Danish and Swedish. These languages contain words that have more than one accepted spelling.

When a language has more than one way of spelling a word, Oracle indexes the word in its basic form. For example in German, the basic form of the *ä* character is *ae*, and so words containing the *ä* character are indexed with *ae* as the substitution.

Oracle also converts query terms to their basic forms before lookup. As a result, users can query words with either spelling.

Enabling Alternate Spelling

You enable alternate spelling by specifying either GERMAN, DANISH, or SWEDISH for the alternate spelling BASIC_LEXER attribute. For example, to enable alternate spelling in German, you can issue the following statements:

```
begin
ctx_ddl.create_preference('GERMAN_LEX', 'BASIC_LEXER');
ctx_ddl.set_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING', 'GERMAN');
end;
```

Disabling Alternate Spelling

To disable alternate spelling, use the CTX_DDL.UNSET_ATTRIBUTE procedure as follows:

```
begin
ctx_ddl.unset_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING');
end;
```

German Alternate Spelling

The German alphabet is the English alphabet plus the additional characters: ä ö ü ß. The following table lists the alternate spelling conventions Oracle Text uses for these characters.

Character	Alternate Spelling Substitution
ä	ae
ü	ue
ö	oe
Ä	AE
Ü	UE
Ö	OE
ß	ss

Danish Alternate Spelling

The Danish alphabet is the Latin alphabet without the *w*, plus the special characters: ø æ å. The following table lists the alternate spelling conventions Oracle Text uses for these characters.

Character	Alternate Spelling Substitution
æ	ae
ø	oe
å	aa
Æ	AE
Ø	OE
Å	AA

Swedish Alternate Spelling

The Swedish alphabet is the English alphabet without the *w*, plus the additional characters: å ä ö. The following table lists the alternate spelling conventions Oracle Text uses for these characters.

Character	Alternate Spelling Substitution
ä	ae
å	aa
ö	oe
Ä	AE
Å	AA
Ö	OE

F

Scoring Algorithm

This appendix describes the scoring algorithm for word queries.You obtain score using the SCORE operator.

Note: This appendix discusses how Oracle calculates score for word queries, which is different from the way it calculates score for ABOUT queries in English.

Scoring Algorithm for Word Queries

To calculate a relevance score for a returned document in a word query, Oracle 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'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
100,000	5
1,000,000	4

The table illustrates that if only one document contained the query term and there were five documents in the set, the term would have to occur 20 times in the document to score 100. Whereas, if there were 1,000,000 documents in the set, the term would have to occur only 4 times in the document to score 100.

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

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 change the score for any given term before and after the DML.

If DML is heavy, you or your Oracle administrator must optimize the index. Perfect relevance ranking is obtained by executing a query right after optimizing the index.

If DML is light, Oracle still gives fairly accurate relevance ranking.

In either case, you or your Oracle administrator must synchronize the index with CTX_DDL.SYNC_INDEX.

G

Views

This appendix lists all of the views provided by Oracle Text. The system provides the following views:

- CTX_CLASSES
- 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_PARAMETERS
- CTX_PENDING
- CTX_PREFERENCES
- CTX_PREFERENCE_VALUES
- CTX_SECTIONS

- CTX_SECTION_GROUPS
- CTX_SERVERS
- CTX_SQES
- CTX_STOPLISTS
- CTX_STOPWORDS
- CTX_SUB_LEXERS
- CTX_THESAURI
- CTX_THES_PHRASES
- CTX_USER_INDEXES
- CTX_USER_INDEX_ERRORS
- CTX_USER_INDEX_OBJECTS
- CTX_USER_INDEX_PARTITIONS
- CTX_USER_INDEX_SETS
- CTX_USER_INDEX_SET_INDEXES
- CTX_USER_INDEX_SUB_LEXERS
- CTX_USER_INDEX_SUB_LEXER_VALS
- CTX_USER_INDEX_VALUES
- CTX_USER_PENDING
- CTX_USER_PREFERENCES
- CTX_USER_PREFERENCE_VALUES
- CTX_USER_SECTIONS
- CTX_USER_SECTION_GROUPS
- CTX_USER_SQES
- CTX_USER_STOPLISTS
- CTX_USER_STOPWORDS
- CTX_USER_SUB_LEXERS
- CTX_USER_THESAURI
- CTX_USER_THES_PHRASES

- CTX_VERSION

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_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_ID	NUMBER	Internal index id.
IDX_OWNER	VARCHAR2(30)	Owner of index.
IDX_NAME	VARCHAR2(30)	Name of index.
IDX_TABLE_OWNER	VARCHAR2(30)	Owner of table.
IDX_TABLE	VARCHAR2(30)	Table name.
IDX_KEY_NAME	VARCHAR2(256)	Primary key column(s).
IDX_TEXT_NAME	VARCHAR2(30)	Text column name.
IDX_DOCID_COUNT	NUMBER	Number of documents indexed.
IDX_STATUS	VARCHAR2(12)	Status.
IDX_LANGUAGE_COLUMN	VARCHAR2(256)	Name of the language column in base table.
IDX_FORMAT_COLUMN	VARCHAR2(256)	Name of the format column in base table.
IDX_CHARSET_COLUMN	VARCHAR2(256)	Name of the charset column in base table.

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 (i.e. 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_TABLE_OWNER	VARCHAR2(30)	Table owner.
IXP_TABLE_NAME	VARCHAR2(30)	Table name.
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, etc.)
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, etc.)
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.
OAT_STATIC	VARCHAR2(1)	Not currently used.
OAT_DATATYPE	VARCHAR2(64)	Attribute datatype
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_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.
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.

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_SERVERS

This view displays `ctxsrv` servers that are currently active. It can be queried only by CTXSYS.

Column Name	Type	Description
SER_NAME	VARCHAR2(60)	Server identifier
SER_STATUS	VARCHAR2(8)	Server status (IDLE, RUN, EXIT)
SER_ADMMBX	VARCHAR2(60)	Admin pipe mailbox name for server.
SER_OOBMBX	VARCHAR2(60)	Out-of-band mailbox name for server.
SER_SESSION	NUMBER	No Longer Used
SER_AUDSID	NUMBER	Server audit session ID
SER_DBID	NUMBER	Server database ID
SER_PROCID	VARCHAR2(10)	No Longer Used
SER_PERSON_MASK	VARCHAR2(30)	Personality mask for server
SER_STARTED_AT	DATE	Date on which server was started
SER_IDLE_TIME	NUMBER	Idle time, in seconds, for server
SER_DB_INSTANCE	VARCHAR2(10)	Database instance ID
SER_MACHINE	VARCHAR2(64)	Name of host machine on which server is running

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	VARCHAR2(2000)	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_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_ID	NUMBER	Internal index id.
IDX_NAME	VARCHAR2(30)	Name of index.
IDX_TYPE	VARCHAR2(30)	Type of index: CONTEXT, CTXCAT, OR CTXRULE
IDX_TABLE_OWNER	VARCHAR2(30)	Owner of table.
IDX_TABLE	VARCHAR2(30)	Table name.
IDX_KEY_NAME	VARCHAR2(256)	Primary key column(s).
IDX_TEXT_NAME	VARCHAR2(30)	Text column name.
IDX_DOCID_COUNT	NUMBER	Number of documents indexed.
IDX_STATUS	VARCHAR2(12)	Status, either INDEXED or INDEXING.
IDX_LANGUAGE_COLUMN	VARCHAR2(256)	Name of the language column of base table.
IDX_FORMAT_COLUMN	VARCHAR2(256)	Name of the format column of base table.
IDX_CHARSET_COLUMN	VARCHAR2(256)	Name of the charset column of base table.

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 (i.e. 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_ID	NUMBER(38)	Index partition id.
IXP_INDEX_NAME	VARCHAR2(30)	Index name.
IXP_INDEX_PARTITION_NAME	VARCHAR2(30)	Index partition name.
IXP_TABLE_OWNER	VARCHAR2(30)	Table owner.
IXP_TABLE_NAME	VARCHAR2(30)	Table name.
IXP_TABLE_PARTITION_NAME	VARCHAR2(30)	Table partition name.
IXP_DOCID_COUNT	NUMBER(38)	Number of documents associated with the index partition.
IXP_STATUS	VARCHAR2(12)	Partition status.

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_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__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 for all system and session SQEs that have been created by the current user. It can be viewed by all users.

Column Name	Type	Description
SQE_OWNER	VARCHAR2(30)	Owner of SQE.
SQE_NAME	VARCHAR2(30)	Name of SQE.
SQE_QUERY	VARCHAR2(2000)	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 thesaurus 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.
VER_CODE	VARCHAR2(9)	The version number of the code linked in to the Oracle shadow process. This column calls out to a trusted callout to fetch the version number for linked-in code. Thus, you can use this column to detect and verify patch releases.

H

Stopword Transformations

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 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 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 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 Expression	Rewritten Expression
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

NOT Transformations

Stopword Expression	Rewritten Expression
non_stopword NOT *stopword*	non_stopword
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

I

English Knowledge Base Category Hierarchy

This appendix provides a list of all the concepts in the knowledge base that serve as categories.

The appendix is divided into six sections, corresponding to the six main branches of the knowledge base:

- Branch 1: science and technology
- Branch 2: business and economics
- Branch 3: government and military
- Branch 4: social environment
- Branch 5: geography
- Branch 6: abstract ideas and concepts

The categories are presented in an inverted-tree hierarchy and within each category, sub-categories are listed in alphabetical order.

> **Note:** This appendix does not contain all the concepts found in the knowledge base. It only contains those concepts that serve as categories (meaning they are parent nodes in the hierarchy).

Branch 1: science and technology

[1] communications

[2] journalism
    [3] broadcast journalism
    [3] photojournalism
    [3] print journalism
        [4] newspapers
[2] public speaking
[2] publishing industry
    [3] desktop publishing
    [3] periodicals
        [4] business publications
    [3] printing
[2] telecommunications industry
    [3] computer networking
        [4] Internet technology
            [5] Internet providers
            [5] Web browsers
            [5] search engines
    [3] data transmission
    [3] fiber optics
    [3] telephone service

[1] formal education

[2] colleges and universities
    [3] academic degrees
    [3] business education
[2] curricula and methods
[2] library science
[2] reference books
[2] schools
[2] teachers and students

[1] hard sciences

[2] aerospace industry
    [3] satellite technology
    [3] space exploration
        [4] Mars exploration
        [4] lunar exploration
        [4] space explorers
        [4] spacecraft and space stations
[2] chemical industry
    [3] chemical adhesives
    [3] chemical dyes
    [3] chemical engineering
    [3] materials technology
        [4] industrial ceramics
        [4] metal industry
            [5] aluminum industry
            [5] metallurgy
            [5] steel industry
        [4] plastics
        [4] rubber
        [4] synthetic textiles
    [3] paints and finishing materials
    [3] pesticides
        [4] fungicides
        [4] herbicides
[2] chemistry
    [3] chemical properties
    [3] chemical reactions
    [3] chemicals
        [4] chemical acids
        [4] chemical elements
        [4] molecular reactivity
        [4] molecular structure
    [3] chemistry tools
        [4] chemical analysis
        [4] chemistry glassware
        [4] purification and isolation of chemicals
    [3] organic chemistry
    [3] theory and physics of chemistry
[2] civil engineering
    [3] building architecture
    [3] construction industry
        [4] building components
            [5] exterior structures
                [6] entryways and extensions
                [6] landscaping
                [6] ornamental architecture
                [6] roofs and towers
                [6] walls
                [6] windows
            [5] interior structures
                [6] building foundations
                [6] building systems
                    [7] electrical systems
                    [7] fireproofing and insulation
                    [7] plumbing
                [6] rooms
        [4] buildings and dwellings

[5] outbuildings
[4] carpentry
[4] construction equipment
[4] construction materials
[5] paneling and composites
[5] surfaces and finishing
[2] computer industry
[3] computer hardware industry
[4] computer components
[5] computer memory
[5] microprocessors
[4] computer peripherals
[5] data storage devices
[4] hand-held computers
[4] laptop computers
[4] mainframes
[4] personal computers
[4] workstations
[3] computer science
[4] artificial intelligence
[3] computer security and data encryption
[4] computer viruses and protection
[3] computer software industry
[4] CAD-CAM
[4] client-server software
[4] computer programming
[5] programming development tools
[5] programming languages
[4] operating systems
[3] computer standards
[3] cyberculture
[3] human-computer interaction
[3] information technology
[4] computer multimedia
[5] computer graphics
[5] computer sound
[5] computer video
[4] databases
[4] document management
[4] natural language processing
[4] spreadsheets
[3] network computing
[3] supercomputing and parallel computing
[3] virtual reality
[2] electrical engineering
[2] electronics
[3] consumer electronics
[4] audio electronics
[4] video electronics
[3] electronic circuits and components
[4] microelectronics
[4] semiconductors and superconductors
[3] radar technology
[2] energy industry
[3] electric power industry
[3] energy sources
[4] alternative energy sources
[4] fossil fuels industry
[5] coal industry
[5] petroleum products industry
[4] nuclear power industry
[2] environment control industries
[3] heating and cooling systems
[3] pest control
[3] waste management
[2] explosives and firearms
[3] chemical explosives
[3] firearm parts and accessories
[3] recreational firearms
[2] geology
[3] geologic formations
[3] geologic substances
[4] mineralogy
[5] gemstones
[5] igneous rocks
[5] metamorphic rocks
[5] sedimentary rocks
[3] hydrology
[3] meteorology
[4] atmospheric science
[4] clouds
[4] storms
[4] weather modification
[4] weather phenomena
[4] winds
[3] mining industry
[3] natural disasters
[3] oceanography
[3] seismology
[3] speleology
[3] vulcanology
[2] inventions
[2] life sciences
[3] biology
[4] biochemistry
[5] biological compounds
[6] amino acids
[6] enzymes

```
                [6] hormones
                    [7] androgens and anabolic
steroids
                    [7] blood sugar hormones
                    [7] corticosteroids
                    [7] estrogens and progestins
                    [7] gonadotropins
                    [7] pituitary hormones
                    [7] thyroid hormones
                [6] lipids and fatty acids
                [6] nucleic acids
                [6] sugars and carbohydrates
                [6] toxins
                [6] vitamins
            [5] cell reproduction
            [5] cell structure and function
            [5] molecular genetics
        [4] botany
            [5] algae
            [5] fungi
            [5] plant diseases
            [5] plant kingdom
                [6] ferns
                [6] flowering plants
                    [7] cacti
                    [7] grasses
                [6] mosses
                [6] trees and shrubs
                    [7] conifers
                    [7] deciduous trees
                    [7] palm trees
            [5] plant physiology
                [6] plant development
                [6] plant parts
        [4] lower life forms
            [5] bacteria
            [5] viruses
        [4] paleontology
            [5] dinosaurs
        [4] physiology
            [5] anatomy
                [6] cardiovascular systems
                [6] digestive systems
                [6] extremities and appendages
                [6] glandular systems
                [6] head and neck
                    [7] ear anatomy
                    [7] eye anatomy
                    [7] mouth and teeth
                [6] immune systems
                    [7] antigens and antibodies
                [6] lymphatic systems
                [6] muscular systems
                [6] nervous systems
                [6] reproductive systems
                [6] respiratory systems
                [6] skeletal systems
                [6] tissue systems
                [6] torso
                [6] urinary systems
            [5] reproduction and development
    [4] populations and vivisystems
        [5] biological evolution
        [5] ecology
            [6] ecological conservation
            [6] environmental pollution
        [5] genetics and heredity
    [4] zoology
        [5] invertebrates
            [6] aquatic sponges
            [6] arthropods
                [7] arachnids
                    [8] mites and ticks
                    [8] scorpions
                    [8] spiders
                [7] crustaceans
                [7] insects
            [6] coral and sea anemones
            [6] jellyfish
            [6] mollusks
                [7] clams, oysters, and
mussels
                [7] octopi and squids
                [7] snails and slugs
            [6] starfish and sea urchins
            [6] worms
        [5] vertebrates
            [6] amphibians
            [6] birds
                [7] birds of prey
                    [8] owls
                [7] game birds
                [7] hummingbirds
                [7] jays, crows, and magpies
                [7] parrots and parakeets
                [7] penguins
                [7] pigeons and doves
                [7] warblers and sparrows
```

```
                    [7] ophthalmic
vasoconstrictors
                [6] otics, ear care drugs
            [5] fluid and electrolyte balance
drugs
                [6] diuretics
            [5] hormonal agents
            [5] immune system drugs
                [6] antitoxins and antivenins
                [6] biological response modifiers
                [6] immune serums
                [6] immunosuppressants
                [6] vaccines and toxoids
            [5] oxytocics
            [5] respiratory drugs
                [6] antihistamines
                [6] bronchodilators
                [6] expectorants and antitussives
            [5] spasmolytics
            [5] topical agents
    [3] health and medicine
        [4] healthcare industry
            [5] healthcare providers and practices
            [5] medical disciplines and
specialties
                [6] cardiology
                [6] dentistry
                [6] dermatology
                [6] geriatrics
                [6] neurology
                [6] obstetrics and gynecology
                [6] oncology
                [6] ophthalmology
                [6] pediatrics
            [5] medical equipment
                [6] artificial limbs and organs
                [6] dressings and supports
            [5] medical equipment manufacturers
            [5] medical facilities
        [4] medical problems
            [5] blood disorders
            [5] cancers and tumors
                [6] carcinogens
            [5] cardiovascular disorders
            [5] developmental disorders
            [5] environment-related afflictions
            [5] gastrointestinal disorders
            [5] genetic and hereditary disorders
            [5] infectious diseases
                [6] communicable diseases
                    [7] sexually transmitted
diseases
            [5] injuries
            [5] medical disabilities
            [5] neurological disorders
            [5] respiratory disorders
            [5] skin conditions
        [4] nutrition
        [4] practice of medicine
            [5] alternative medicine
            [5] medical diagnosis
                [6] medical imaging
            [5] medical personnel
            [5] medical procedures
                [6] physical therapy
                [6] surgical procedures
                    [7] cosmetic surgery
        [4] veterinary medicine
[2] machinery
    [3] machine components
[2] mathematics
    [3] algebra
        [4] linear algebra
        [4] modern algebra
    [3] arithmetic
        [4] elementary algebra
    [3] calculus
    [3] geometry
        [4] mathematical topology
        [4] plane geometry
        [4] trigonometry
    [3] math tools
    [3] mathematical analysis
    [3] mathematical foundations
        [4] number theory
        [4] set theory
        [4] symbolic logic
    [3] statistics
[2] mechanical engineering
[2] physics
    [3] acoustics
    [3] cosmology
        [4] astronomy
            [5] celestial bodies
                [6] celestial stars
                [6] comets
                [6] constellations
                [6] galaxies
```

[6] moons
[6] nebulae
[6] planets
[5] celestial phenomena
[3] electricity and magnetism
[3] motion physics
[3] nuclear physics
[4] subatomic particles
[3] optical technology
[4] holography
[4] laser technology
[5] high-energy lasers
[5] low-energy lasers
[3] thermodynamics
[2] robotics
[2] textiles
[2] tools and hardware
[3] cements and glues
[3] hand and power tools
[4] chisels
[4] drills and bits
[4] gauges and calipers
[4] hammers
[4] machine tools
[4] planes and sanders
[4] pliers and clamps
[4] screwdrivers
[4] shovels
[4] trowels
[4] wrenches
[3] knots

[1] social sciences

[2] anthropology
[3] cultural identities
[4] Native Americans
[3] cultural studies
[4] ancient cultures
[3] customs and practices
[2] archeology
[3] ages and periods
[3] prehistoric humanoids
[2] history
[3] U.S. history
[4] slavery in the U.S.
[3] ancient Rome
[4] Roman emperors
[3] ancient history
[3] biographies
[3] historical eras
[2] human sexuality
[3] homosexuality
[3] pornography
[3] prostitution
[3] sexual issues
[2] linguistics
[3] descriptive linguistics
[4] grammar
[5] parts of speech
[4] phonetics and phonology
[3] historical linguistics
[3] languages
[3] linguistic theories
[3] rhetoric and figures of speech
[3] sociolinguistics
[4] dialects and accents
[3] writing and mechanics
[4] punctuation and diacritics
[4] writing systems
[2] psychology
[3] abnormal psychology
[4] anxiety disorders
[4] childhood onset disorders
[4] cognitive disorders
[4] dissociative disorders
[4] eating disorders
[4] impulse control disorders
[4] mood disorders
[4] personality disorders
[4] phobias
[4] psychosomatic disorders
[4] psychotic disorders
[4] somatoform disorders
[4] substance related disorders
[3] behaviorist psychology
[3] cognitive psychology
[3] developmental psychology
[3] experimental psychology
[3] humanistic psychology
[3] neuropsychology
[3] perceptual psychology
[3] psychiatry
[3] psychoanalytic psychology
[3] psychological states and behaviors
[3] psychological therapy
[3] psychological tools and techniques
[3] sleep psychology

[4] sleep disorders
[2] sociology
[3] demographics
[3] social identities
[4] gender studies
[4] senior citizens
[3] social movements and institutions
[3] social structures

[1] transportation

[2] aviation
[3] aircraft
[3] airlines
[3] airports
[3] avionics
[2] freight and shipping
[3] package delivery industry
[3] trucking industry
[2] ground transportation
[3] animal powered transportation
[3] automotive industry
[4] automobiles
[4] automotive engineering
[5] automotive parts
[5] internal combustion engines
[4] automotive sales
[4] automotive service and repair
[4] car rentals
[4] motorcycles
[4] trucks and buses
[3] human powered vehicles
[3] rail transportation
[4] subways
[4] trains
[3] roadways and driving
[2] marine transportation
[3] boats and ships
[3] seamanship
[3] waterways
[2] travel industry
[3] hotels and lodging
[3] tourism
[4] cruise lines
[4] places of interest
[4] resorts and spas

Branch 2: business and economics

[1] business services industry

[1] commerce and trade

[2] electronic commerce
[2] general commerce
[2] international trade and finance
[2] mail-order industry
[2] retail trade industry
[3] convenience stores
[3] department stores
[3] discount stores
[3] supermarkets
[2] wholesale trade industry

[1] corporate business

[2] business enterprise
[3] entrepreneurship
[2] business fundamentals
[2] consulting industry
[2] corporate finance
[3] accountancy
[2] corporate management
[2] corporate practices
[2] diversified companies
[2] human resources
[3] employment agencies
[2] office products
[2] quality control
[3] customer support
[2] research and development
[2] sales and marketing
[3] advertising industry

[1] economics

[1] financial institutions

[2] banking industry
[2] insurance industry
[2] real-estate industry

[1] financial investments

[2] commodities market
    [3] money
        [4] currency market
    [3] precious metals market
[2] general investment
[2] personal finance
    [3] retirement investments
[2] securities market
    [3] bond market
    [3] mutual funds
    [3] stock market

[1] financial lending

[2] credit cards

[1] industrial business

[2] industrial engineering
    [3] production methods
[2] industrialists and financiers
[2] manufacturing
    [3] industrial goods manufacturing

[1] public sector industry

[1] taxes and tariffs

[1] work force

[2] organized labor

Branch 3: government and military

[1] government

[2] county government
[2] forms and philosophies of government
[2] government actions
[2] government bodies and institutions
    [3] executive branch
        [4] U.S. presidents
        [4] executive cabinet
    [3] judiciary branch
        [4] Supreme Court
            [5] chief justices
    [3] legislative branch
        [4] house of representatives
        [4] senate
[2] government officials
    [3] royalty and aristocracy
    [3] statesmanship
[2] government programs
    [3] social programs
        [4] welfare
[2] international relations
    [3] Cold War
    [3] diplomacy
    [3] immigration
[2] law
    [3] business law
    [3] courts
    [3] crimes and offenses
        [4] controlled substances
            [5] substance abuse
        [4] criminals
        [4] organized crime
    [3] law enforcement
    [3] law firms
    [3] law systems
        [4] constitutional law
    [3] legal bodies
    [3] legal customs and formalities
    [3] legal judgments
    [3] legal proceedings
    [3] prisons and punishments
[2] municipal government
    [3] municipal infrastructure
    [3] urban areas

[4] urban phenomena
[4] urban structures
[2] politics
[3] civil rights
[3] elections and campaigns
[3] political activities
[3] political advocacy
[4] animal rights
[4] consumer advocacy
[3] political parties
[3] political principles and philosophies
[4] utopias
[3] political scandals
[3] revolution and subversion
[4] terrorism
[2] postal communications
[2] public facilities
[2] state government

[1] military

[2] air force
[2] armored clothing
[2] army
[2] cryptography
[2] military honors
[2] military intelligence
[2] military leaders
[2] military ranks
[3] army, air force, and marine ranks
[3] navy and coast guard ranks
[2] military wars
[3] American Civil War
[3] American Revolution
[3] World War I
[3] World War II
[3] warfare
[2] military weaponry
[3] bombs and mines
[3] chemical and biological warfare
[3] military aircraft
[3] missiles, rockets, and torpedoes
[3] nuclear weaponry
[3] space-based weapons
[2] navy
[3] warships
[2] service academies

Branch 4: social environment

[1] belief systems

[2] folklore
[2] mythology
[3] Celtic mythology
[3] Egyptian mythology
[3] Greek mythology
[3] Japanese mythology
[3] Mesopotamian and Sumerian mythology
[3] Norse and Germanic mythology
[3] Roman mythology
[3] South and Central American mythology
[3] mythological beings
[3] myths and legends
[2] paranormal phenomena
[3] astrology
[3] occult
[3] superstitions
[2] philosophy
[3] epistemology
[3] ethics and aesthetics
[3] metaphysics
[3] philosophical logic
[3] schools of philosophy
[2] religion
[3] God and divinity
[3] doctrines and practices
[3] history of religion
[3] religious institutions and structures
[3] sacred texts and objects
[4] Bible
[4] liturgical garments
[3] world religions
[4] Christianity
[5] Christian denominations
[5] Christian heresies
[5] Christian theology
[5] Mormonism
[5] Roman Catholicism
[6] popes
[6] religious orders
[5] evangelism
[5] protestant reformation
[4] Islam
[4] Judaism
[4] eastern religions
[5] Buddhism

[5] Hinduism
[6] Hindu deities

[1] clothing and appearance

[2] clothing
[3] clothing accessories
[4] belts
[4] functional accessories
[4] gloves
[3] fabrics
[4] laces
[4] leather and fur
[3] footwear
[3] garment parts
[4] garment fasteners
[4] garment trim
[3] headgear
[4] hats
[4] helmets
[3] laundry
[3] neckwear
[3] outer garments
[4] dresses
[4] formalwear
[4] jackets
[4] pants
[4] shirts
[4] skirts
[4] sporting wear
[4] sweaters
[3] sewing
[3] undergarments
[4] deshabille
[4] hosiery
[4] lingerie
[4] men's underwear
[2] cosmetics
[3] facial hair
[3] hair styling
[2] fashion industry
[3] supermodels
[2] grooming
[3] grooming aids
[2] jewelry

[1] emergency services

[2] emergency dispatch
[2] emergency medical services
[2] fire prevention and suppression
[2] hazardous material control
[2] heavy rescue

[1] family

[2] death and burial
[3] funeral industry
[2] divorce
[2] infancy
[2] kinship and ancestry
[2] marriage
[2] pregnancy
[3] contraception
[2] upbringing

[1] food and agriculture

[2] agribusiness
[2] agricultural equipment
[2] agricultural technology
[3] soil management
[4] fertilizers
[2] aquaculture
[2] cereals
[2] condiments
[2] crop grain
[2] dairy products
[3] cheeses
[2] drinking and dining
[3] alcoholic beverages
[4] beers
[4] liqueurs
[4] liquors
[4] mixed drinks
[4] wines
[5] wineries
[3] cooking
[3] meals and dishes
[4] sandwiches
[3] non-alcoholic beverages
[4] coffee
[4] soft drinks
[4] tea
[2] farming
[2] fats and oils
[3] butter and margarine
[2] food and drink industry

[3] foodservice industry
[3] meat packing industry
[2] forestry
[3] forest products
[2] fruits and vegetables
[3] legumes
[2] leavening agents
[2] mariculture
[2] meats
[3] beef
[3] pate and sausages
[3] pork
[3] poultry
[2] nuts and seeds
[2] pasta
[2] prepared foods
[3] breads
[3] candies
[3] crackers
[3] desserts
[4] cakes
[4] cookies
[4] pies
[3] pastries
[3] sauces
[3] soups and stews
[2] ranching
[2] seafood
[2] spices and flavorings
[3] sweeteners

[1] housekeeping and butlery

[1] housewares

[2] beds
[2] candles
[2] carpets and rugs
[2] cases, cabinets, and chests
[2] chairs and sofas
[2] curtains, drapes, and screens
[2] functional wares
[3] cleaning supplies
[2] home appliances
[2] kitchenware
[3] cookers
[3] fine china
[3] glassware
[3] kitchen appliances
[3] kitchen utensils
[4] cutting utensils
[3] pots and pans
[3] serving containers
[3] tableware
[2] lamps
[2] linen
[2] mirrors
[2] ornamental objects
[2] stationery
[2] stools and stands
[2] tables and desks
[2] timepieces

[1] leisure and recreation

[2] arts and entertainment
[3] broadcast media
[4] radio
[5] amateur radio
[4] television
[3] cartoons, comic books, and superheroes
[3] cinema
[4] movie stars
[4] movie tools and techniques
[4] movies
[3] entertainments and spectacles
[4] entertainers
[3] humor and satire
[3] literature
[4] children's literature
[4] literary criticism
[4] literary devices and techniques
[4] poetry
[5] classical poetry
[4] prose
[5] fiction
[6] horror fiction
[6] mystery fiction
[4] styles and schools of literature
[3] performing arts
[4] dance
[5] ballet
[5] choreography
[5] folk dances
[5] modern dance
[4] drama
[5] dramatic structure

[5] stagecraft
[4] music
[5] blues music
[5] classical music
[5] composition types
[5] folk music
[5] jazz music
[5] music industry
[5] musical instruments
[6] keyboard instruments
[6] percussion instruments
[6] string instruments
[6] wind instruments
[7] brass instruments
[7] woodwinds
[5] opera and vocal
[5] popular music and dance
[5] world music
[3] science fiction
[3] visual arts
[4] art galleries and museums
[4] artistic painting
[5] painting tools and techniques
[5] styles and schools of art
[4] graphic arts
[4] photography
[5] cameras
[5] photographic lenses
[5] photographic processes
[5] photographic techniques
[5] photographic tools
[4] sculpture
[5] sculpture tools and techniques

[2] crafts
[2] games
[3] indoor games
[4] board games
[4] card games
[4] video games
[3] outdoor games
[2] gaming industry
[3] gambling
[2] gardening
[2] hobbies
[3] coin collecting
[3] stamp collecting
[2] outdoor recreation
[3] hunting and fishing
[2] pets
[2] restaurant industry
[2] sports
[3] Olympics
[3] aquatic sports
[4] canoeing, kayaking, and rafting
[4] swimming and diving
[4] yachting
[3] baseball
[3] basketball
[3] bicycling
[3] bowling
[3] boxing
[3] equestrian events
[4] horse racing
[4] rodeo
[3] fantasy sports
[3] fitness and health
[4] fitness equipment
[3] football
[3] golf
[3] gymnastics
[3] martial arts
[3] motor sports
[4] Formula I racing
[4] Indy car racing
[4] NASCAR racing
[4] drag racing
[4] motorcycle racing
[4] off-road racing
[3] soccer
[3] sports equipment
[3] tennis
[3] track and field
[3] winter sports
[4] hockey
[4] ice skating
[4] skiing
[2] tobacco industry
[2] toys

Branch 5: geography

[1] cartography

[2] explorers

[1] physical geography

[2] bodies of water
- [3] lakes
- [3] oceans
- [3] rivers

[2] land forms
- [3] coastlands
- [3] continents
- [3] deserts
- [3] highlands
- [3] islands
- [3] lowlands
- [3] mountains
- [3] wetlands

[1] political geography

[2] Africa
- [3] Central Africa
  - [4] Angola
  - [4] Burundi
  - [4] Central African Republic
  - [4] Congo
  - [4] Gabon
  - [4] Kenya
  - [4] Malawi
  - [4] Rwanda
  - [4] Tanzania
  - [4] Uganda
  - [4] Zaire
  - [4] Zambia
- [3] North Africa
  - [4] Algeria
  - [4] Chad
  - [4] Djibouti
  - [4] Egypt
  - [4] Ethiopia
  - [4] Libya
  - [4] Morocco
  - [4] Somalia
  - [4] Sudan
  - [4] Tunisia
- [3] Southern Africa
  - [4] Botswana
  - [4] Lesotho
  - [4] Mozambique
  - [4] Namibia
  - [4] South Africa
  - [4] Swaziland
  - [4] Zimbabwe
- [3] West Africa
  - [4] Benin
  - [4] Burkina Faso
  - [4] Cameroon
  - [4] Equatorial Guinea
  - [4] Gambia
  - [4] Ghana
  - [4] Guinea
  - [4] Guinea-Bissau
  - [4] Ivory Coast
  - [4] Liberia
  - [4] Mali
  - [4] Mauritania
  - [4] Niger
  - [4] Nigeria
  - [4] Sao Tome and Principe
  - [4] Senegal
  - [4] Sierra Leone
  - [4] Togo

[2] Antarctica

[2] Arctic
- [3] Greenland
- [3] Iceland

[2] Asia
- [3] Central Asia
  - [4] Afghanistan
  - [4] Bangladesh
  - [4] Bhutan
  - [4] India
  - [4] Kazakhstan
  - [4] Kyrgyzstan
  - [4] Nepal
  - [4] Pakistan
  - [4] Tajikstan
  - [4] Turkmenistan
  - [4] Uzbekistan
- [3] East Asia
  - [4] China
  - [4] Hong Kong
  - [4] Japan

[4] Macao
[4] Mongolia
[4] North Korea
[4] South Korea
[4] Taiwan
[3] Southeast Asia
[4] Brunei
[4] Cambodia
[4] Indonesia
[4] Laos
[4] Malaysia
[4] Myanmar
[4] Papua New Guinea
[4] Philippines
[4] Singapore
[4] Thailand
[4] Vietnam
[2] Atlantic area
[3] Azores
[3] Bermuda
[3] Canary Islands
[3] Cape Verde
[3] Falkland Islands
[2] Caribbean
[3] Antigua and Barbuda
[3] Bahamas
[3] Barbados
[3] Cuba
[3] Dominica
[3] Dominican Republic
[3] Grenada
[3] Haiti
[3] Jamaica
[3] Netherlands Antilles
[3] Puerto Rico
[3] Trinidad and Tobago
[2] Central America
[3] Belize
[3] Costa Rica
[3] El Salvador
[3] Guatemala
[3] Honduras
[3] Nicaragua
[3] Panama
[2] Europe
[3] Eastern Europe
[4] Albania
[4] Armenia
[4] Azerbaijan
[4] Belarus
[4] Bulgaria
[4] Czech Republic
[4] Czechoslovakia
[4] Estonia
[4] Greece
[4] Hungary
[4] Latvia
[4] Lithuania
[4] Moldava
[4] Poland
[4] Republic of Georgia
[4] Romania
[4] Russia
[5] Siberia
[4] Slovakia
[4] Soviet Union
[4] Ukraine
[4] Yugoslavia
[5] Bosnia and Herzegovina
[5] Croatia
[5] Macedonia
[5] Montenegro
[5] Serbia
[5] Slovenia
[3] Western Europe
[4] Austria
[4] Belgium
[4] Denmark
[4] Faeroe Island
[4] Finland
[4] France
[4] Germany
[4] Iberia
[5] Andorra
[5] Portugal
[5] Spain
[4] Ireland
[4] Italy
[4] Liechtenstein
[4] Luxembourg
[4] Monaco
[4] Netherlands
[4] Norway
[4] San Marino
[4] Sweden
[4] Switzerland
[4] United Kingdom
[5] England

```
            [5] Northern Ireland
            [5] Scotland
            [5] Wales
    [2] Indian Ocean area
        [3] Comoros
        [3] Madagascar
        [3] Maldives
        [3] Mauritius
        [3] Seychelles
        [3] Sri Lanka
    [2] Mediterranean
        [3] Corsica
        [3] Cyprus
        [3] Malta
        [3] Sardinia
    [2] Middle East
        [3] Bahrain
        [3] Iran
        [3] Iraq
        [3] Israel
        [3] Jordan
        [3] Kuwait
        [3] Lebanon
        [3] Oman
        [3] Palestine
        [3] Qatar
        [3] Saudi Arabia
        [3] Socotra
        [3] Syria
        [3] Turkey
        [3] United Arab Emirates
        [3] Yemen
    [2] North America
        [3] Canada
        [3] Mexico
        [3] United States
            [4] Alabama
            [4] Alaska
            [4] Arizona
            [4] Arkansas
            [4] California
            [4] Colorado
            [4] Delaware
            [4] Florida
            [4] Georgia
            [4] Hawaii
            [4] Idaho
            [4] Illinois
            [4] Indiana
            [4] Iowa
            [4] Kansas
            [4] Kentucky
            [4] Louisiana
            [4] Maryland
            [4] Michigan
            [4] Minnesota
            [4] Mississippi
            [4] Missouri
            [4] Montana
            [4] Nebraska
            [4] Nevada
            [4] New England
                [5] Connecticut
                [5] Maine
                [5] Massachusetts
                [5] New Hampshire
                [5] Rhode Island
                [5] Vermont
            [4] New Jersey
            [4] New Mexico
            [4] New York
            [4] North Carolina
            [4] North Dakota
            [4] Ohio
            [4] Oklahoma
            [4] Oregon
            [4] Pennsylvania
            [4] South Carolina
            [4] South Dakota
            [4] Tennessee
            [4] Texas
            [4] Utah
            [4] Virginia
            [4] Washington
            [4] Washington D.C.
            [4] West Virginia
            [4] Wisconsin
            [4] Wyoming
    [2] Pacific area
        [3] American Samoa
        [3] Australia
            [4] Tasmania
        [3] Cook Islands
        [3] Fiji
        [3] French Polynesia
        [3] Guam
        [3] Kiribati
        [3] Mariana Islands
```

[3] Marshall Islands
[3] Micronesia
[3] Nauru
[3] New Caledonia
[3] New Zealand
[3] Palau
[3] Solomon Islands
[3] Tonga
[3] Tuvalu
[3] Vanuatu
[3] Western Samoa
[2] South America
[3] Argentina
[3] Bolivia
[3] Brazil
[3] Chile
[3] Colombia
[3] Ecuador
[3] French Guiana
[3] Guyana
[3] Paraguay
[3] Peru
[3] Suriname
[3] Uruguay
[3] Venezuela

Branch 6: abstract ideas and concepts

[1] dynamic relations

[2] activity
[3] attempts
[4] achievement
[4] difficulty
[4] ease
[4] extemporaneousness
[4] failure
[4] preparation
[4] success
[3] inertia
[3] motion
[4] agitation
[4] directional movement
[5] ascent
[5] convergence
[5] departure
[5] descent
[5] divergence
[5] entrance
[5] inward motion
[5] jumps
[5] motions around
[5] outward motion
[5] progression
[5] withdrawal
[4] forceful motions
[5] friction
[5] pulls
[5] pushes
[5] throws
[4] haste
[4] slowness
[4] transporting
[3] rest
[3] violence
[2] change
[3] exchanges
[3] gradual change
[3] major change
[3] reversion
[2] time
[3] future
[3] longevity

[3] past
[3] regularity of time
[3] relative age
[4] stages of development
[3] simultaneity
[3] time measurement
[4] instants
[3] timeliness
[4] earliness
[4] lateness
[3] transience

[1] human life and activity

[2] communication
[3] announcements
[3] conversation
[3] declarations
[3] disclosure
[3] identifiers
[3] implication
[3] obscene language
[3] representation
[4] interpretation
[3] secrecy
[3] shyness
[3] speech
[3] styles of expression
[4] boasting
[4] clarity
[4] eloquence
[4] intelligibility
[4] nonsense
[4] plain speech
[4] wordiness
[2] feelings and sensations
[3] calmness
[3] composure
[3] emotions
[4] anger
[4] contentment
[4] courage
[4] cowardice
[4] happiness
[4] humiliation
[4] ill humor
[4] insolence
[4] nervousness
[4] pickiness
[4] regret
[4] relief
[4] sadness
[4] vanity
[3] excitement
[3] five senses
[4] audiences
[4] hearing
[5] faintness of sound
[5] loudness
[5] silence
[5] sound
[6] cries
[6] dissonant sound
[6] harmonious sound
[6] harsh sound
[6] repeated sounds
[4] sight
[5] appearance
[5] fading
[5] visibility
[4] smelling
[5] odors
[4] tasting
[5] flavor
[6] sweetness
[4] touching
[3] numbness
[3] pleasure
[3] suffering
[2] gender
[2] intellect
[3] cleverness
[3] foolishness
[3] ignorance
[3] intelligence and wisdom
[3] intuition
[3] knowledge
[3] learning
[3] teaching
[3] thinking
[4] conclusion
[5] discovery
[5] evidence
[5] rebuttal
[4] consideration
[5] analysis
[5] questioning
[5] tests

[4] faith
[5] ideology
[5] sanctimony
[4] judgment
[4] rationality
[4] skepticism
[4] sophistry
[4] speculation

[2] social attitude, custom
[3] behavior
[4] approval
[4] courtesy
[4] criticism
[4] cruelty
[4] flattery
[4] forgiveness
[4] friendliness
[4] generosity
[4] gratitude
[4] hatred
[4] jealousy
[4] kindness
[4] love
[5] adoration
[4] respect
[4] rudeness
[4] ruthlessness
[4] stinginess
[4] sympathy
[3] morality and ethics
[4] evil
[4] goodness
[4] moral action
[5] asceticism
[5] decency
[5] deception
[5] integrity
[5] lewdness
[5] self-indulgence
[4] moral consequences
[5] allegation
[5] entitlement
[5] excuses
[5] punishment
[5] reparation
[4] moral states
[5] fairness
[5] guilt
[5] innocence
[5] partiality
[4] responsibility
[3] reputation
[4] acclaim
[4] notoriety
[3] social activities
[4] enjoyment
[4] monotony
[3] social conventions
[4] conventionalism
[4] formality
[4] trends
[3] social transactions
[4] debt
[4] offers
[4] payments
[4] petitions
[4] promises and contracts

[2] states of mind
[3] anticipation
[4] fear
[4] frustration
[4] hopefulness
[4] hopelessness
[4] prediction
[4] surprise
[4] warnings
[3] boredom
[3] broad-mindedness
[3] carelessness
[3] caution
[3] confusion
[3] creativity
[3] curiosity
[3] forgetfulness
[3] patience
[3] prejudice
[3] remembering
[3] seriousness

[2] volition
[3] assent
[3] choices
[4] denial
[3] decidedness
[3] dissent
[3] eagerness
[3] enticement
[3] evasion
[4] abandonment

[4] escape
[3] impulses
[3] indecision
[3] indifference
[3] inevitability
[3] motivation
[3] obstinacy
[3] tendency

[1] potential relations

[2] ability, power
[3] competence, expertise
[3] energy, vigor
[3] ineptness
[3] productivity
[3] provision
[3] strength
[3] weakness
[2] conflict
[3] attacks
[3] competition
[3] crises
[3] retaliation
[2] control
[3] anarchy
[3] command
[4] cancelations
[4] delegation
[4] permission
[4] prohibiting
[3] defiance
[3] influence
[3] leadership
[3] modes of authority
[4] confinement
[4] constraint
[4] discipline
[4] freedom
[4] leniency
[4] liberation
[3] obedience
[3] regulation
[3] servility
[2] possession
[3] giving
[3] keeping
[3] losing
[3] receiving
[3] sharing
[3] taking
[2] possibility
[3] chance
[3] falseness
[3] truth
[2] purpose
[3] abuse
[3] depletion
[3] obsolescence
[2] support
[3] cooperation
[3] mediation
[3] neutrality
[3] peace
[3] protection
[3] sanctuary
[3] security

[1] relation

[2] agreement
[2] cause and effect
[3] causation
[3] result
[2] difference
[2] examples
[2] relevance
[2] similarity
[3] duplication
[2] uniformity
[2] variety

[1] static relations

[2] amounts
[3] fewness
[3] fragmentation
[3] large quantities
[3] majority
[3] mass quantity
[3] minority
[3] numbers
[3] quantity modification
[4] combination
[4] connection
[4] decrease
[4] increase
[4] remainders

[4] separation
[3] required quantity
[4] deficiency
[4] excess
[4] sufficiency
[3] wholeness
[4] omission
[4] thoroughness
[2] existence
[3] creation
[3] life
[2] form
[3] defects
[3] effervescence
[3] physical qualities
[4] brightness and color
[5] color
[6] variegation
[5] colorlessness
[5] darkness
[5] lighting
[6] opaqueness
[6] transparency
[4] dryness
[4] fragility
[4] heaviness
[4] mass and weight measurement
[4] moisture
[4] pliancy
[4] rigidity
[4] softness
[4] temperature
[5] coldness
[5] heat
[4] texture
[5] fluids
[5] gaseousness
[5] jaggedness
[5] powderiness
[5] semiliquidity
[5] smoothness
[4] weightlessness
[3] shape
[4] angularity
[4] circularity
[4] curvature
[4] roundness
[4] straightness
[3] symmetry
[3] tangibility
[3] topological form
[4] concavity
[4] convexity
[4] covering
[4] folds
[4] openings
[2] nonexistence
[3] death
[3] destruction
[2] quality
[3] badness
[3] beauty
[3] cleanness
[3] complexity
[3] correctness
[3] deterioration
[3] dirtiness
[3] good quality
[3] improvement
[3] mediocrity
[3] mistakes
[3] normality
[3] perfection
[3] remedy
[3] simplicity
[3] stability
[4] resistance to change
[3] strangeness
[3] ugliness
[3] value
[2] range
[3] areas
[4] area measurement
[4] regions
[4] storage
[4] volume measurement
[3] arrangement
[4] locations
[5] anteriors
[5] compass directions
[5] exteriors
[5] interiors
[5] left side
[5] posteriors
[5] right side
[5] topsides
[5] undersides
[4] positions

```
            [5] disorder
            [5] groups
                [6] dispersion
                [6] exclusion
                [6] inclusion
                [6] itemization
                [6] seclusion
                [6] togetherness
            [5] hierarchical relationships
                [6] downgrades
                [6] ranks
                [6] upgrades
            [5] sequence
                [6] beginnings
                [6] continuation
                [6] ends
                [6] middles
                [6] preludes
    [3] boundaries
    [3] dimension
        [4] contraction
        [4] depth
        [4] expansion
        [4] flatness
        [4] height
        [4] largeness
        [4] length
        [4] linear measurement
        [4] narrowness
        [4] shallowness
        [4] shortness
        [4] slopes
        [4] smallness
        [4] steepness
        [4] thickness
    [3] essence
    [3] generalization
    [3] nearness
    [3] obstruction
    [3] remoteness
    [3] removal
    [3] significance
    [3] trivialness
    [3] uniqueness
    [3] ways and methods
```

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