Oracle8*i interMedia* Text

Reference

Release 2 (8.1.6)

December 1999

Part No. A77063-01

ORACLE

Oracle8*i* *inter*Media Text Reference, Release 2 (8.1.6)

Part No. A77063-01

Primary Author: Colin McGregor

Contributors: Shamim Alpha, Steve Buxton, Chung-Ho Chen, Yun Cheng, Paul Dixon, Mohammad Faisal, Elena Huang, Jie Bai, Garret Kaminaga, Jacqueline Kud, Bryn Llewellyn, Wesley Lin, Kavi Mahesh, Yasuhiro Matsuda, Gerda Shank, and Steve Yang.

Contents

2 SQL Commands

3 Indexing

4 Query Operators

11 CTX_THES Package

12 Executables

C Supported Filter Formats

D Loading Examples

E Supplied Stoplists

F Alternate Spelling Conventions

G Scoring Algorithm

H Views

I Stopword Transformations

J Knowledge Base - Category Hierarchy

Index

Send Us Your Comments

Oracle8*i interM*edia Text Reference, Release 2 (8.1.6)

Part No. A77063-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. 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 about this manual?

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

- E-mail: infodev@us.oracle.com
- Fax: (650) 506-7228
- Postal service:
  Oracle Corporation
  Attn: Oracle8i Server Documentation
  500 Oracle Parkway
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, and telephone number below.

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

Preface

This manual provides reference information for Oracle8*i* *inter*Media Text. Use it as a reference for creating interMedia Text indexes, for issuing Text queries, for presenting documents, and for using the interMedia Text PL/SQL packages.

Audience

This manual is intended for a interMedia Text application developer or a system administrator responsible for maintaining the interMedia Text system.

Prerequisites

This document assumes that you have 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 *Oracle8 Concepts.* The chapter is a comprehensive introduction to the concepts and terminology used throughout Oracle documentation.

Related Publications

For more information about *inter*Media Text, see:

- *Oracle8i interMedia Text Migration*

For more information about Oracle8*i*, see:

- *Oracle8i Concepts*
- *Oracle8i Administrator's Guide*
- *Oracle8i Utilities*
- *Oracle8i Designing and Tuning for Performance*
- *Oracle8i SQL Reference*
- *Oracle8i Reference*
- *Oracle8i Application Developer's Guide - Fundamentals*

For more information about PL/SQL, see:

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

How This Manual Is Organized

See the table of contents.

Type Conventions

This manual adheres to the following type conventions:

Type	Meaning
UPPERCASE	Uppercase letters indicate Oracle commands, standard database objects and constants, and standard Oracle PL/SQL procedures.
italics	Italics indicate query terms in CONTAINS queries. Italics also indicate emphasis.
`monospace`	Monospace type indicate example SQL*Plus commands and example PL/SQL code. Type in the command or code exactly as it appears.

Customer Support

You can reach Oracle Worldwide Customer Support 24 hours a day.

In the USA: **1.650.506.1500**

In Europe: **+ 44.344.860.160**

Please be prepared to supply the following information:

- your CSI number (This helps Oracle Corporation track problems for each customer)
- the release numbers of the Oracle Server and associated products
- the operating system name and version number
- details of error numbers and descriptions (Write down the exact errors you encounter)
- a description of the problem
- a description of the changes made to the system

Your Comments Are Welcome

Please use the Send Us Your Comments form in this document to convey your comments to us. You can also contact us at:

Documentation Manager
Oracle8i Server Documentation
Oracle Corporation

500 Oracle Parkway
Redwood Shores, California 94065
Phone: 1.650.506.7000 FAX: 1.650.506.7228

1

Introduction to *inter*Media Text

This chapter introduces the main features of Oracle8*i* *inter*Media Text. It is provided to help you get started with indexing, querying, and document presentation.

The following topics are covered:

- What is Oracle8i interMedia Text?
- Building Your Text Application
- interMedia Text Users and Roles
- Loading Documents
- Indexing
- Index Maintenance
- Querying
- Document Presentation and Highlighting

What is Oracle8i interMedia Text?

You use Oracle8i interMedia Text to build a text query application. Oracle8i interMedia Text provides search, retrieval, and viewing capabilities for text. In addition, interMedia Text provides concept searching and theme analysis of English language documents.

To index and query, you use standard SQL with a domain index of type `context`. You can also use the supplied interMedia Text PL/SQL packages for advanced features such as document presentation and thesaurus maintenance.

Building Your Text Application

This chapter introduces the main features of *inter*Media Text to help you design and build your query application. The examples in this chapter are based on the out-of-box default behavior.

To build a query application, you must understand the users and roles associated with interMedia text. These are described in the next section, "interMedia Text Users and Roles".

The general steps for building a query application are the following:

1. Load the documents. See "Loading Documents" in this chapter.
2. Index the documents. See "Indexing" in this chapter.
3. Issue queries. See "Querying" in this chapter.
4. Present the documents that satisfy a query. See "Document Presentation and Highlighting" in this chapter.

The sections that follow give a general overview of these tasks.

Code Samples Directory

Code samples are included in your Oracle8*i* *inter*Media Text installation. You can view these samples by pointing your browser to the following file:

```
$ORACLE_HOME/ctx/sample/api/index.html
```

These code samples illustrate the SQL and PL/SQL interface for *inter*Media Text. The examples are intended for programmers already familiar with *inter*Media Text concepts.

The examples contain `.sql` files you execute in SQL*Plus and `.sh` files you execute at the command line.

Typically, you run several files in sequence to illustrate a given feature. You can access each sequence via links from an HTML page which provides further explanation where appropriate.

*inter*Media Text Users and Roles

While any user can create an *inter*Media Text index and issue a CONTAINS query, Oracle8*i* *inter*Media Text provides the CTXSYS user for administration and the CTXAPP role for application developers.

CTXSYS User

The CTXSYS user is created at install time. You administrate interMedia Text users as this user.

CTXSYS can do the following

- modify system-defined preferences
- drop and modify other user preferences
- call procedures in the CTX_ADM PL/SQL package to start servers and set system-parameters
- start a `ctxsrv` server
- query all system-defined views
- perform all the tasks of a user with the CTXAPP role

CTXAPP Role

The CTXAPP role is a system-defined role that enables users to do the following:

- create and delete interMedia Text preferences
- use the interMedia Text PL/SQL packages

Any user can create an interMedia Text index and issue a Text query. The CTXAPP role allows users create preferences and use the PL/SQL packages.

Loading Documents

The default indexing behavior expects documents loaded in a text column.

Note: Even though the system expects documents to be loaded in a text column, you can also store your documents in other ways, including the file system and as a URL.

For more information about data storage, see "Datastore Objects" in Chapter 3.

Supported Column Types

By default, the system expects your documents to be loaded in a text column. Your text column can be VARCHAR2, CLOB, BLOB, CHAR or BFILE.

Note: Storing data in the deprecated column types of LONG and LONG RAW is supported only for migrating Oracle7 systems to Oracle8.

The column types NCLOB, DATE and NUMBER cannot be indexed.

Supported Document Formats

Because the system can index most document formats including HTML, PDF, Microsoft Word, and plain text, you can load any of these document types into the text column.

See Also: For more information about the supported document formats, see Appendix C, "Supported Filter Formats".

Loading Methods

INSERT Statement

You can use the SQL INSERT statement to load documents to a table.

The following example creates a table with two columns, `id` and `text`, using CREATE TABLE. The example populates the table with the INSERT statement. This

example makes the `id` column the primary key, which is the required constraint for a Text table. The `text` column is VARCHAR2:

```
create table docs (id number primary key, text varchar2(80));
```

To populate this table, 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');
```

Loading Text Data from File-System

In addition to the INSERT statement, Oracle enables you to load text data (this includes documents, pointers to documents, and URLs) into a table from your file-system using other automated methods, including

- SQL*Loader
- ctxload executable
- DBMS_LOB.LOADFROMFILE() PL/SQL procedure to load LOBs from BFILEs
- Oracle Call Interface

> **See Also:** For loading examples, including how to use SQL*Loader, see Appendix D, "Loading Examples".
>
> To learn more about ctxload, see "`ctxload`" in Chapter 11.
>
> For more information about the DBMS_LOB package, see *Oracle8i Supplied PL/SQL Packages Reference*.
>
> For more information about working with LOBs, see the *Oracle8i Application Developer's Guide - Large Objects (LOBs)*.
>
> For more information about Oracle Call Interface, see *Oracle Call Interface Programmer's Guide*

Indexing

Once your text data is loaded in a table and the table contains a primary key, you can run the command to create a Text index.

For example, the following command creates a Text index called `myindex` on the `text` column in the `docs` table:

```
create index myindex on docs(text) indextype is ctxsys.context;
```

General Defaults for All Languages

When you use CREATE INDEX without explicitly specifying parameters, the system does the following for all languages by default:

- Assumes that the text to be indexed is stored directly in a text column. The text column can be of type CLOB, BLOB, BFILE, VARCHAR2, or CHAR. The columns types LONG and LONG RAW are supported for migrating Oracle7 systems to Oracle8i. The column types NCLOB, DATE and NUMBER cannot be indexed
- Detects the column type and uses filtering for binary column types. Most document formats are supported for filtering. If your column is plain text, the system does not use filtering.

  Note: For document filtering to work correctly in your system, you must ensure that your environment is set up correctly to support the Inso filter.

  To learn more about configuring your environment to use the Inso filter, see "About Inso Filtering Technology" in Appendix C.

- Assumes the language of text to index is the language you specify in your database setup.
- Uses the default stoplist for the language you specify in your database setup. Stoplists identify the words that the system ignores during indexing.
- Enables fuzzy and stemming queries for your language, if this feature is available for your language.

You can always change the default indexing behavior by creating your own preferences and specifying these custom preferences in the parameter string of CREATE INDEX.

See Also: To learn more about creating your own custom preferences, see Chapter 3, "Indexing".

See also CTX_DDL.CREATE_PREFERENCE in Chapter 7.

To learn more about using CREATE INDEX, see its specification in Chapter 2.

Language Specific Defaults

American and English Language Defaults

In addition to the general defaults, the system enables the following option for American and English language text:

- Indexes document theme information. When theme information is indexed, ABOUT queries are more precise

Danish Language Defaults

In addition to the general defaults, the system enables the following options for Danish text:

- Alternate spelling

Dutch Language Defaults

In addition to the general defaults, the system enables the following options for Dutch text:

- Composite indexing
- Alternate spelling

Finnish, Norwegian, and Swedish Language Defaults

In addition to the general defaults, the system enables the following option for Finnish, Norwegian, and Swedish text:

- Alternate spelling

German and German DIN Language Defaults

In addition to the general defaults, the system enables the following options for German text:

- Case-sensitive indexing

- Composite indexing
- Alternate spelling

> **See Also:** To learn more about these options, see BASIC_LEXER in Chapter 3.

Index Maintenance

Index maintenance is necessary after your application inserts, updates, or deletes documents in your base table.

If your base table is static, that is, you do no updating, inserting or deleting of documents after your initial index, you do not need to maintain your index.

However, if you perform DML (inserts, updates, or deletes) on your base table, you must update your index. You can synchronize your index manually with ALTER INDEX. You can also run the `ctxsrv` server in the background which synchronizes the index automatically at regular intervals.

> **See Also:** For more information about synchronizing the index, see ALTER INDEX in Chapter 2.
>
> For more information about `ctxsrv`, see "`ctxsrv`" in Chapter 11.

Querying

You issue Text queries using the CONTAINS operator in a SELECT statement. With CONTAINS, you can issue two types of queries:

- word query
- ABOUT query

You can also optimize queries for better response time. The following sections give an overview of these query scenarios.

Word Query Example

A word query is a query on the exact word or phrase you enter between the single quotes in the CONTAINS operator.

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

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

In your query expression, you can use text operators such as AND and OR to achieve different results. You can also add structured predicates to the WHERE clause.

> **See Also:** For more information about the different operators you can use in queries, see Chapter 4, "Query Operators".

You can count the hits to a query using count(*), or CTX_QUERY.COUNT_HITS.

ABOUT Query Example

In all languages, ABOUT queries increases the number of relevant documents returned by a query.

In English, ABOUT queries can use the theme component of the index, which is created by default. As such, this operator returns documents based on the concepts of your query, not only the exact word or phrase you specify.

For example, the following query finds all the documents in the *text* column that are about the subject *politics*, not just the documents that contain the word *politics*:

```
SELECT SCORE(1) title from news
```

```
WHERE CONTAINS(text, 'about(politics)', 1) > 0;
```

See Also: For more information about the ABOUT operator, see ABOUT in Chapter 4, "Query Operators".

Optimizing Query for Response Time

You can optimize any CONTAINS query (word or ABOUT) for response time in order to retrieve the highest ranking hits in a result set in the shortest time possible. Optimizing for response time is useful in a web-based search application.

To optimize for response time, use the FIRST_ROWS hint. For example, the following PL/SQL block uses a cursor to retrieve the first 20 hits of a query and uses the FIRST_ROWS hint to optimize the response time:

```
declare
cursor c is
select /*+ FIRST_ROWS */ pk, score(1), col from ctx_tab
           where contains(txt_col, 'oracle', 1) > 0 order by score(1) desc;
begin
for i in c
loop
insert into t_s values(i.pk, i.col);
exit when c%rowcount > 21;
end loop;
end;
```

See Also: For more information on optimizing queries, see Appendix A, "Query Tuning"

Other Query Features

In your query application, you can use other query features such as section searching. The following table lists some of these features and shows where to look in this book for more information.

Feature	Where to Find More Information
Section Searching	Chapter 7, "CTX_DDL Package" for defining sections. WITHIN Operator in Chapter 4 for searching within sections.
Proximity Searching	NEAR (;) Operator in Chapter 4.

Feature	Where to Find More Information
Stem and Fuzzy Searching	stem ($) and fuzzy (?) operators in Chapter 4 for issuing queries. "Wordlist Object" in Chapter 3 for setting the options for your language.
Thesaural Queries	Chapter 4, "Query Operators" for using thesaurus operators in queries. Thesaurus operators include SYN, BT, and NT. Chapter 11, "CTX_THES Package" for browsing and altering a thesaurus. "`ctxload`" in Chapter 11 for loading a thesaurus.
Case Sensitive Searching Base Letter Conversion Word Decompounding (German and Dutch) Alternate Spelling (German, Dutch, and Swedish)	"Lexer Objects" in Chapter 3 for enabling these features.
Optimizing Queries	Appendix A, "Query Tuning"
Query Explain Plan	CTX_QUERY.EXPLAIN procedure in Chapter 10.
Hierarchical Query Feedback	CTX_QUERY.HFEEDBACK procedure in Chapter 10.

Document Presentation and Highlighting

Typically, a Text query application allows the user to view the documents returned by a query. The user selects a document from the hitlist and then your application presents the document in some form.

With *inter*Media Text, you can render a document in different ways. For example, you can present documents with query terms highlighted. Highlighted query terms can be either the words of a word query or the themes of an ABOUT query in English.

You can also obtain theme information from documents with the CTX_DOC PL/SQL package.

Table 1–1 describes the different output you can obtain and which procedure to use to obtain each type:

Table 1–1

Output	Procedure
Highlighted document, plain text version	CTX_DOC.MARKUP
Highlighted document, HTML version	CTX_DOC.MARKUP
Highlight offset information for plain text version	CTX_DOC.HIGHLIGHT
Highlight offset information for HTML version	CTX_DOC.HIGHLIGHT
Plain text version, no highlights	CTX_DOC.FILTER
HTML version of document, no highlights	CTX_DOC.FILTER
Theme summaries and Gist of document.	CTX_DOC.GIST
List of themes in document.	CTX_DOC.THEMES

See Also: For more information about these procedures, see Chapter 8, "CTX_DOC Package".

2

SQL Commands

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

The following commands are described in this chapter:

- ALTER INDEX
- CONTAINS
- CREATE INDEX
- DROP INDEX
- SCORE

ALTER INDEX

> **Note:** This section describes the ALTER INDEX command as it pertains to managing a Text domain index.
>
> For a complete description of the ALTER INDEX command, see *Oracle8i SQL Reference*.

Purpose

Use ALTER INDEX to perform the following maintenance tasks for a Text index:

- Rename the index. See RENAME Syntax.
- Rebuild the index using different preferences. See REBUILD Syntax.
- Resume a failed index operation (creation / optimization). See REBUILD Syntax.
- Process DML in batch (synchronize). See REBUILD Syntax.
- Optimize the index. See REBUILD Syntax.
- Add stopwords to the index. See REBUILD Syntax.
- Add sections and stop sections to the index. See REBUILD Syntax.

RENAME Syntax

Use the following syntax to rename an index:

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

schema.index_name
Specify the name of the index to be renamed.

new_index_name
Specify the new name for schema.index. The new_index_name parameter can be no more than 25 characters. If you specify a name longer than 25 characters, Oracle returns an error and the renamed index is no longer valid.

Note: When new_index_name has more than 25 characters and less than 30 characters, 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 command and then recreate and drop index_name.

REBUILD Syntax

The following syntax is used to rebuild the index, 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 [online] [parameters (paramstring)];
```

[online]
Optionally specify the online parameter for non-blocking operation, which allows the index to be queried during an ALTER INDEX synchronize or optimize operation. You cannot specify online for replace, resume, or when adding stopwords or stop sections.

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]

              |    resume [memory memsize]
              |    optimize [fast | full [maxtime (time | unlimited)]
              |    sync [memory memsize]
              |    add stopword word
              |    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.

> **See Also:** For more information about creating and setting preferences, including information about system-defined preferences, see Chapter 3, "Indexing".

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

optimize [fast | full [maxtime (*time* | unlimited)]
Optimizes the index. Specify either fast or full optimization.

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 (garbage collection.)

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, which ever 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.

sync [memory *memsize*]
Synchronizes the index. You can optionally specify the amount of runtime memory to use with memsize.

add stopword *word*
Dynamically adds a stopword word to the index.

The index is *not* rebuilt by this command.

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 command.

> **See Also:** For more information on add section constraints, see "Add Section Constraints" in this section.

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:** For more information about visible and invisible field sections, see CTX_DDL.ADD_FIELD_SECTION.

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 command.

> **See Also:** For more information on add section constraints, see "Add Section Constraints" in this section.

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 command.

> **See Also:** For more information on add section constraints, see "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 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 command.

> **See Also:** For more information on add section constraints, see "Add Section Constraints" in this section.

Examples

Resuming Failed Index

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

```
ALTER INDEX newsindex rebuild parameters('resume memory 2M');
```

Rebuilding an Index

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

```
ALTER INDEX newsindex rebuild parameters('replace stoplist new_stop');
```

Fast Optimization

The following command optimizes `newsindex` in fast mode:

```
ALTER INDEX newsindex rebuild parameters('optimize fast');
```

Full Optimization

To specify an optimization operation to last for three hours (180 minutes), issue the following command:

```
ALTER INDEX newsindex rebuild parameters('optimize full maxtime 180');
```

To optimize the entire index without regard to time, issue the following command:

```
ALTER INDEX newsindex rebuild parameters('optimize full maxtime unlimited');
```

To optimize the entire index and to allow queries to be issued during the optimization, issue the following command:

```
ALTER INDEX newsindex rebuild online parameters('optimize full maxtime
unlimited');
```

Synchronizing the Index

The following example synchronizes the index with a runtime memory of 2 megabytes:

```
ALTER INDEX newsindex rebuild PARAMETERS('sync memory 2M');
```

Adding a Zone Section

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

```
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 command:

```
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 command:

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

Notes

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 memory improves indexing performance since there is less I/O 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 if you want to track indexing progress or when run-time memory is scarce.

Viewing Index Errors

You can view index errors with iMT views. View errors on your indexes with CTX_USER_INDEX_ERRORS. View errors on all indexes as CTXSYS with the CTX_INDEX_ERRORS.

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;
```

To clear the view, you can issue:

```
delete from CTX_USER_INDEX_ERRORS;
```

Viewing Pending DML

With the CTX_USER_PENDING view, you can view newly updated, deleted, or inserted rows requiring index synchronization. For example, to view pending DML on all your indexes, issue the following statement:

```
select pnd_index_name, pnd_rowid, to_char(pnd_timestamp, 'dd-mon-yyyy
hh24:mi:ss') timestamp from ctx_user_pending;
```

This statement gives output in the form:

```
PND_INDEX_NAME                 PND_ROWID          TIMESTAMP
------------------------------ ------------------ --------------------
MYINDEX                        AAADXnAABAAAS3SAAC 06-oct-1999 15:56:50
```

Add Section Constraints

Before altering the index section information, the new section is checked 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

Related Topics

CTX_DDL.CREATE_PREFERENCE in Chapter 7.

CTX_DDL.CREATE_STOPLIST in Chapter 7.

CTX_DDL.CREATE_SECTION_GROUP in Chapter 7.

CREATE INDEX

DROP INDEX

DROP INDEX

> **Note:** This section describes the DROP INDEX command as it pertains to dropping a Text domain index.
>
> For a complete description of the DROP INDEX command, see *Oracle8i 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

CONTAINS

Purpose

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.

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 the query expression that defines your search in column.

> **See Also:** For more information about the Text operators you can use in query expressions, see Chapter 4, "Query Operators".

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 operator must reference the label value as in the following example.

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

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. Since the index contains tokens from multiple languages, such a query can return documents in several languages. To limit you query to a given language, use a structured clause on the language column.

Related Topics

SCORE

Appendix A, "Query Tuning"

CREATE INDEX

> **Note:** This section describes the CREATE INDEX command as it pertains to creating a Text domain index.
>
> For a complete description of the CREATE INDEX command, see *Oracle8i SQL Reference*.

Purpose

Use CREATE INDEX to create an *inter*Media Text index. An *inter*Media Text index is an Oracle domain index of type context created using the extensible indexing framework.

You must create an iMT index to issue CONTAINS queries.

Syntax

```
CREATE INDEX [schema.]index on [schema.]table(column) INDEXTYPE IS
ctxsys.context [PARAMETERS(paramstring)] [PARALLEL N];
```

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

[schema.]table(column)
Specify the name of the table and column to index. The table must have a primary key constraint. This is needed mainly for identifying the documents for document services. Composite primary keys are supported, up to 16 columns.

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

> **Note:** Indexing the deprecated column types LONG and LONG RAW is supported for the process of migrating Oracle7 systems to Oracle8*i*.

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.

Composite indexes are not supported; you must specify only one column in the column list.

PARALLEL N

Optionally specify with N the parallel degree for parallel indexing. You must have a partitioned text table to use parallel indexing with iMT. Each parallel indexer works on its own partition, so N must not exceed the number of partitions.

The initialization parameters JOB_QUEUE_PROCESSES and JOB_QUEUE_INTERVAL must also be set in the `init.ora` file to use parallel indexing. The parallel degree N you specify must be less than JOB_QUEUE_PROCESSES.

A complete example of how to create an index in parallel is given in the Examples section.

> **See Also:** For more information about these initialization parameters, see Oracle8i Reference and Oracle8i Administrator's Guide.

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.

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

For more information about these defaults, see "Default Index Parameters" in Chapter 3.

---

datastore *datastore_pref*
Specify the name of your datastore preference. See Datastore Objects in Chapter 3.

filter *filter_pref*
Specify the name of your filter preference. See Filter Objects in Chapter 3.

charset column *charset_column_name*
Specify the name of the character set column. The charset column must be in the same table as the text column and it must be CHAR, VARCHAR, or VARCHAR2 type. Use this column to specify the document character set for conversion to the database character set. The value is case-insensitive, but must be an NLS character set string such as JA16EUC.

When the document is plain-text or HTML, the INSO_FILTER object and CHARSET filter object 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.

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 either TEXT or BINARY. 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.

lexer *lexer_pref*
Specify the name of your lexer or multi-lexer preference. See Lexer Objects in Chapter 3.

language column *language_column_name*
Specify the name of the language column when using a multi-lexer preference. See MULTI_LEXER in Chapter 3.

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. See Wordlist Object in Chapter 3.

storage *storage_pref*
Specify the name of your storage preference for the Text index. See Storage Objects in Chapter 3.

stoplist *stoplist*
Specify the name of your stoplist. See CTX_DDL.CREATE_STOPLIST in Chapter 7.

section group *section_group*
Specify the name of your section group. See CTX_DDL.CREATE_SECTION_GROUP in Chapter 7.

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 for memsize must be between 1M and the value specified for max_index_memory in the CTX_PARAMETERS view. 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 since there is less I/O 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 if you want to track indexing progress or 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.

Examples

Creating Index Using Default Preferences

The following example creates a Text 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 3.
>
> Also refer to "Indexing" in Chapter 1.

Creating Index with Custom Preferences

The following example creates a Text 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_DDLCREATE_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 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 3.

Parallel Indexing

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

1. Set job queue parameters in the Oracle initialization file to accommodate immediate parallel indexing:

```
job_queue_processes = 8
job_queue_interval = 4
```

The JOB_QUEUE_PROCESSES parameter must be greater than or equal to your parallel degree.

This example starts the indexing job within four seconds of issuing the CREATE INDEX command. If you are not worried about the immediacy of the indexing operation, you can set a longer JOB_QUEUE_INTERVAL in seconds.

> **See Also:** For more information about these initialization parameters, see Oracle8i Reference and Oracle8i Administrator's Guide

2. Create a partitioned text table. The following example creates a hash-partitioned table using values in the pk column as the hash key:

```
create table mypart_tab(pk number primary key, text varchar2(80))
PARTITION BY HASH (pk) PARTITIONS 8;
```

3. Create the index with a parallel degree. This example uses a parallel degree of 3.

```
create index myindex on mypart_tab(pk) indextype is ctxsys.context parallel 3;
```

Notes

The issuing user does not need the CTXAPP role to create an index. If the user has Oracle grants to create a b-tree index on the column, then this user has sufficient permission to create a Text index. The issuing owner, table owner, and index owner can all be different users, which is the standard behavior for regular b-tree indexes.

Index Errors

You can view index errors with iMT views. View errors on your indexes with CTX_USER_INDEX_ERRORS. View errors on all indexes as CTXSYS with the CTX_INDEX_ERRORS.

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;
```

To clear the view of errors, you can issue:

```
delete from CTX_USER_INDEX_ERRORS;
```

Related Topics

CTX_DDL.CREATE_PREFERENCE in Chapter 7.

CTX_DDL.CREATE_STOPLIST in Chapter 7.

CTX_DDL.CREATE_SECTION_GROUP in Chapter 7.

ALTER INDEX

DROP INDEX

SCORE

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

Syntax

```
SCORE(label NUMBER)
```

label
Specify a number to identify the score produced by the query.

Notes

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

Example

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 G, "Scoring Algorithm"

3

Indexing

This chapter describes the index preference objects, section group types, and stoplists you can use to create your interMedia 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	Description
Datastore	How are your documents stored?
Filter	How can the documents be converted to plaintext?
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 the options you have for setting each preference. You enable an option by creating a preference with one of the objects 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 object and specify `mydatastore` as the datastore preference in the parameter string of CREATE INDEX.

Creating Preferences

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

To create a stoplists, use CTX_DDL.CREATE_STOPLIST.

To create section groups, use CTX_DDL.CREATE_SECTION_GROUP and specify a section group type.

Datastore Objects

Use the datastore objects to specify how your text is stored. To create a data storage preference, you must use one of the following objects:

Object	Use When
DIRECT_DATASTORE	Data is stored internally in the text column. Each row is indexed as a single document
DETAIL_DATASTORE	Data is stored internally in the text column. Document consists of one or more rows in a detail table, with header information stored in a master table.
FILE_DATASTORE	Data is stored externally in operating system files. Filenames 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) 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 object for text stored directly in the text column, one document per row. DIRECT_DATASTORE has no attributes.

DIRECT_DATASTORE 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-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');
```

DETAIL_DATASTORE

Use the DETAIL_DATASTORE object for text stored directly in the database in detail tables, with the textkey 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 name of detail table (OWNER.TABLE if necessary)
detail_key	Specify name of detail table foreign key column(s)
detail_lineno	Specify name of detail table sequence column.
detail_text	Specify name of detail table text column.

Example Master/Detail Tables

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

Master Table 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
body	CHAR(1)	Dummy column to specify in CREATE INDEX.

Detail Table 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	CLOB	Document text.

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 above. 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;
```

Index 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 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 object 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 location of text files that are stored externally in a file system.

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 *path* is not used to specify a path for external files, Oracle requires the path to 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 object 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. Non-printable 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 not supported.

URL_DATASTORE Attributes

URL_DATASTORE has the following attributes:

Attribute	Attribute Values
timeout	Specify timeout in seconds. The valid range is 15 to 3600 seconds. The default is 30.
maxthreads	Specify maximum number of threads that can be running simultaneously. Use a number between 1and 1024. Default is 8.
urlsize	Specify maximum length of URL string in bytes. Use number between 32 and 65535. Defaults to 256.
maxurls	Specify maximum size of URL buffer. Use a number between 32 and 65535. Defaults to 256.
maxdocsize	Specify maximum document size. Use a number between 256 and 2,147,483,647 bytes (2 gigabytes). Defaults to 2,000,000.
http_proxy	Specify host name of http proxy server. Optionally specify port number with a colon in the form hostname:port.
ftp_proxy	Specify host name of ftp proxy server. Optionally specify port number with a colon in the form hostname:port.
no_proxy	Specify 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 values specified for maxurls and urlsize, when multiplied, cannot exceed 5,000,000.

In other words, the maximum size of the memory buffer (maxurls * urlsize) for the URL object 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.

maxdocsize
Specify the maximum size, in bytes, that the URL data store 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 *inter*Media Text is installed. You can optionally specify 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.

ftp_proxy
Specify the fully-qualified name of the host machine that serves as the FTP proxy (gateway) for the machine on which *inter*Media 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 to which the http_proxy, no_proxy, 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 USER_DATASTORE object 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 attribute:

Attribute	Attribute Value
procedure	Specify the name of 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. Can be either CLOB, BLOB, or VARCHAR2. The default is CLOB.

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 the following parameters 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 the type defined in 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
- procedure cannot be a safe callout or call a safe callout

Editing Procedure after Indexing If you change or edit the stored procedure, indexes based upon it will not be notified, so you must manually recreate such indexes. That is, if the stored procedure makes use of other columns, and those column values change, the row will not be re-indexed. The row is only re-indexed when the indexed column changes.

output_type
Specify the datatype of the second argument to procedure. You can use either CLOB, BLOB or VARCHAR2.

USER_DATASTORE 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
  offset number := 1;
begin
  for c1 in (select author, title, text from articles
              where rowid = rid)
  loop
    append_varchar_to_lob(tlob, c1.title, offset);
    append_varchar_to_lob(tlob, 'by '||c1.author, offset);
    dbms_lob.append(tlob, c1.text);
  end loop;
end;
```

This procedure takes in a rowid and a temporary clob locator, and concatenates all the articles columns into the temporary clob. Assume that a helper procedure `append_varchar_to_lob` does the concatenation.

Because only ctxsys-owned stored procedures 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 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:

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

When `appowner` creates the index on `articles(text)` using this preference, the indexing operation sees author and title in the document text.

NESTED_DATASTORE

Use the nested datastore 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 nor 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 which contains the text of the line. This is like DETAIL_TEXT in detail datastore. This attribute is required. The text attribute can be of any indexable type except LONG or LONG RAW. 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, since 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 lineno and 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.

```
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 creates a storage table `myntab` for the nested table.

It then 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('mtds','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 tab
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;
-- should get back document 1, since it has dog in line 1 and mat in line 2.
```

Filter Objects

Use the filter objects 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 objects:

Filter Preference Object	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 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.

The CHARSET_FILTER has the following attribute:

Attribute	Attribute Value
charset	Specify the NLS name of source character set. 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: For more information about the supported NLS character sets, see *Oracle8i National Language Support Guide.*

Indexing Mixed-Character Set Columns

A mixed character-set column is one that stores documents of different character sets. For example, a Japanese text column might store documents in JA16EUC and JA16SJIS character sets.

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, the source character set is assumed to be 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 Japanese documents in EUC and name the character set:

```
insert into hdocs values(1, 'text', 'JA16EUC', '/docs/tekusuto.euc');
insert in hdocs values (2, 'text', ''JA16SJIS', '/docs/tekusuto.sjs');
```

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. Use it for indexing single and mixed format columns. The INSO_FILTER has no attributes.

> **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 C, "Supported Filter Formats".

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 string 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. On the other hand, 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 able, 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');
```

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 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 USER_FILTER, except that in this case there is no Japanese character-set auto-detection.

See Also: USER_FILTER in this section.

Plain-Text Indexing and the INSO_FILTER

Oracle does not recommend using INSO_FILTER to index text documents.

If your table contains text documents entirely, use the INSO_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.

If, however, you use the INSO_FILTER to index non-binary documents (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 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
- Or the database character-set is neither of the ones listed above and the document is in the WE8ISO8859P1 character set.

NULL_FILTER

Use the NULL_FILTER object 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 do *not* use the INSO filter. Instead, 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 indexing HTML documents, see "Section Group Types" in this chapter.

USER_FILTER

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

Attribute	Attribute Values
command	*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 go 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 the 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 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');
```

Lexer Objects

Use the lexer preference to specify the language of the text to be indexed. To create a lexer object, you must use one of the following objects:

Object	Description
BASIC_LEXER	Lexer used for extracting tokens from text in languages, such as English and most western European languages that use single-byte character sets.
MULTI_LEXER	Lexer used for indexing tables containing documents of different languages.
CHINESE_VGRAM_LEXER	Lexer used for extracting tokens from Chinese text.
JAPANESE_VGRAM_LEXER	Lexer used for extracting tokens from Japanese text.
KOREAN_LEXER	Lexer used for extracting tokens from Korean text.

BASIC_LEXER

Use the BASIC_LEXER object to identify tokens for creating Text indexes for English and all other supported single-byte languages.

The BASIC_LEXER is also used to enable base-letter conversion, composite word indexing, case-sensitive indexing and alternate spelling for single-byte languages that have extended character sets.

In English, you can use the BASIC_LEXER to enable theme indexing.

> **Note:** Any changes made to tokens before Text indexing (e.g. removing of characters, base-letter conversion) are also performed on the query terms in a Text query. This ensures that the query terms match the form of the tokens in the Text index.

The BASIC_LEXER supports all single-byte character sets plus UTF8.

BASIC_LEXER has the following attributes:

Attribute	Attribute Values
continuation	*characters* (string)
numgroup	*characters* (string)
numjoin	*characters* (string)
printjoins	*characters* (string)
punctuations	*characters* (string)
skipjoins	*characters* (string)
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_themes	YES (enabled)
	NO (disabled)
index_text	YES (enabled)
	NO (disabled)
theme_language	AUTO (default)
	ENGLISH
alternate_spelling	GERMAN (German alternate spelling)
	DANISH (Danish alternate spelling)
	SWEDISH (Swedish alternate spelling)
	NONE (No alternate spelling)

Note: The BASIC_LEXER object attributes that use character strings can contain multiple characters. Each character in the string serves as a distinct character for that type of attribute.

For example, if the string '*_.-' is specified for the printjoins attribute, each individual character ('*', '_', '.', and '-') in the string is treated as a joining character that is included in the index entry for a token in which the character occurs.

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 numgroup characters 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 NLS 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 the BASIC_LEXER object.

---

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 only removed if 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 end 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 / 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 / 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 server as paragraph delimiters for sentence / 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, etc.) 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 converts the tokens in Text index entries to all uppercase or stores the tokens exactly as they appear in the text. The default is NO (tokens converted to all uppercase).

Note: Oracle ensures Text 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).

> **Note:** Base-letter indexing is always off when composite word indexing is set for GERMAN or DUTCH.
>
> Case-sensitivity is always on when composite is set for GERMAN.
>
> Case-sensitivity can be on or off when composite is set for DUTCH.

index_themes
Specify YES to index theme information in English. 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.

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. The default is AUTO, which instructs the system to set this parameter according to the language of the environment.

> **Note:** In release 8.1.6, the system supports generating themes in English (ENGLISH) only.

index_text
Specify YES to index word information. The index_themes and index_text attributes cannot both be NO.

The default is YES.

alternate_spelling
Specify either GERMAN, DANISH, or SWEDISH to enable alternate spelling in one of these languages. 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 F, "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 above, issue the following statement:

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

MULTI_LEXER

Use this 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 create a multi-lexer preference with CTX_DDL.CREATE_PREFERENCE and then add language-specific lexers to the multi-lexer preference with CTX_DDL.ADD_SUB_LEXER. You must also have a language column in your base table to index multi-language tables. You specify the language column when you index with CREATE INDEX. See the example.

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_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 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. Also assume that the language column is expressed in ISO 639-2, so we have to 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 string 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. Since the index contains tokens from multiple languages, 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 object 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
- AL24UTFFSS

- UTF8

JAPANESE_VGRAM_LEXER

The JAPANESE_VGRAM_LEXER object 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

KOREAN_LEXER

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

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

- KO16KSC5601
- UTF8

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

Attribute	Attribute Values
verb	Specify TRUE or FALSE to index verb. Default is TRUE.
adjective	Specify TRUE or FALSE to index adjective. 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. Default is TRUE.
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 hanja to hanggeul. Default is TRUE.

Limitations

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

Wordlist Object

Use the wordlist preference to enable the advanced query options such as stemming and fuzzy matching for your language. To create a wordlist preference, you must use BASIC_WORDLIST, which is the only object available.

BASIC_WORDLIST

Use BASIC_WORDLIST object to enable stemming and fuzzy matching for Text indexes.

> **See Also:** For more information about the stem and fuzzy operators, see Chapter 4, "Query Operators".

BASIC_WORDLIST has the following attributes:

Table 3–1

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

Table 3–1

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. Setting fuzzy score means scores below this number are not produced. Default is 60.
fuzzy_numresults	Specify the maximum number of fuzzy expansions. Use a number between 0 and 5000. 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.

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 $ 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.

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.

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.

---

fuzzy_score
Specify a default lower limit of fuzzy score. Specify a number between 0 and 80. Setting fuzzy score means scores below this number are not produced. 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. Default is 100.

Setting a fuzzy expansion limits the expansion to a certain 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 segment during index flushes than with substring index off. Oracle recommends making

available double the usual rollback during create index or decreasing index memory.

BASIC_WORDLIST Example

The following example enables stem and fuzzy 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');
```

Storage Objects

Use the storage preference to specify tablespace and creation parameters for tables associated with a Text index. The system provides a single storage object called BASIC_STORAGE:

Object	Description
BASIC_STORAGE	Indexing object used to specify the tablespace and creation parameters for the database tables and indexes that constitute a Text index.

BASIC_STORAGE

The BASIC_STORAGE object 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, do not specify an index organized table clause.

> **See Also:** For more information about how to specify CREATE TABLE and CREATE INDEX clauses, see their command syntax specification in *Oracle8i SQL Reference*.

BASIC_STORAGE has the following attributes:

BASIC_STORAGE 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.

BASIC_STORAGE Attribute	Attribute Value
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.
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.
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 tablespace 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 string of CREATE INDEX.

To create a section group, you can specify one of the following group types with CTX_DDL.CREATE_SECTION_GROUP:

Section Group Preference	Description
NULL_SECTION_GROUP	This is the default. Use this group type when you define no sections or when you define *only* SENTENCE or PARAGRAPH sections.
BASIC_SECTION_GROUP	Use this group type for defining sections where the start and end tags are of the form <A> and </A>.
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.
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.
NEWS_SECTION_GROUP	Use this group for defining sections in newsgroup formatted documents according to RFC 1036.

Section Group Examples

HTML Documents

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;
```

You can optionally add sections to this group using CTX_DDL.ADD_SECTION. 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');
```

XML Documents

The following command 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 CTX_DDL.ADD_SECTION. 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 command 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');
```

Stoplists

By default, the system indexes text using the system-supplied stoplist that corresponds to the language of your database setup. 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.

Oracle8i interMedia 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 E, "Supplied Stoplists".

Creating Stoplists

You can create your own stoplists using CTX_DLL.CREATE_STOPLIST.

When you create your own stoplist, you must specify it in the parameter string 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 recreate your index for the changes to take effect.

Dynamic Addition of Stopwords

You can *add* stopwords dynamically to a stoplist, default or custom, with ALTER INDEX. When you add a stopword dynamically, you need not re-index since the word immediately becomes 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 and re-index.

System-Defined Preferences

When you install *inter*Media Text, some indexing preferences are created. You can use these preferences in the parameter string of CREATE INDEX or define your own.

The default index parameters, described in the next section, 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" in this chapter.

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 object. It is used 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 object.

CTXSYS.URL_DATASTORE

This preference uses the URL_DATASTORE object.

Filter

CTXSYS.NULL_FILTER

This preference uses the INSO_FILTER object.

CTXSYS.INSO_FILTER

This preference uses the INSO_FILTER object.

Lexer

CTXSYS.DEFAULT_LEXER

This preference defaults to the lexer required for the language you specify during your database setup. 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 enabled. That is, document theme information is indexed for American and English language.

Danish Language Settings If your language is Norwegian, Swedish, or Finnish, 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)
- alternate spelling (alternate_spelling 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_LEXER. All attributes for the KOREAN_LEXER are enabled.

Simplified Chinese Language Settings If your language is Simplified 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 in this chapter.

CTXSYS.BASIC_LEXER

This preference uses the BASIC_LEXER.

Section Group

CTXSYS.NULL_SECTION_GROUP

This preference uses the NULL_SECTION_GROUP object.

CTXSYS.HTML_SECTION_GROUP

This preference uses the HTML_SECTION_GROUP object.

CTXSYS.AUTO_SECTION_GROUP

This preference uses the AUTO_SECTION_GROUP object.

Stoplist

CTXSYS.DEFAULT_STOPLIST

This stoplist preference defaults to the stoplist of the language specified during your database setup.

> **See Also:** For a complete list of the stop words in the supplied stoplists, see Appendix E, "Supplied Stoplists".

CTXSYS.EMPTY_STOPLIST

This stoplist has no words.

Storage

CTXSYS.DEFAULT_STORAGE

This storage preference uses the BASIC_STORAGE object.

Wordlist

CTXSYS.DEFAULT_WORDLIST

This preference uses the language stemmer for the language specified during your database setup. If your language is not listed in Table 3–1, this preference defaults to the NULL stemmer and the GENERIC fuzzy matching attribute.

System Parameters

General

When you install *inter*Media 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 which can be specified in the parameter string 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	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.

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

Default Index Parameters

The following default parameters are used when you do not specify preferences in the parameter string of CREATE 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 string of CREATE INDEX.	CTXSYS.DEFAULT_DATASTORE
DEFAULT_FILTER_FILE	No filter preference specified in parameter string 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 string 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 string 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 string of CREATE INDEX, and when either of the following conditions is true: ■ your datastore preference uses URL_DATASTORE or ■ when your filter preference uses INSO_FILTER.	CTXSYS.HTML_SECTION_GROUP
DEFAULT_SECTION_TEXT	No section group specified in parameter string 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 string of CREATE INDEX.	CTXSYS.DEFAULT_STORAGE

System Parameter	Used When	Default Value
DEFAULT_LEXER	No lexer preference specified in parameter string of CREATE INDEX.	CTXSYS.DEFAULT_LEXER
DEFAULT_STOPLIST	No stoplist specified in parameter string of CREATE INDEX.	CTXSYS.DEFAULT_STOPLIST
DEFAULT_WORDLIST	No wordlist preference specified in parameter string of CREATE INDEX.	CTXSYS.DEFAULT_WORDLIST

Viewing Default Values

You can view system defaults with CTX_PARAMETERS view.

Changing Default Values

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

4

Query Operators

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

- 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:** For more information, see "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 to an ABOUT query with the most relevant document receiving the highest score.

English Behavior

In English, 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 match your query word or phrase.

The word or phrase specified in your ABOUT query does not have to exactly match the themes stored in the index. Oracle automatically normalizes the word or phrase before performing lookup in the index.

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.

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)'
```

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.

Notes

If an index contains only theme information, an ABOUT operator and operand must be included in a query on the text column or else Oracle returns an error.

Oracle ignores any query operators that are included in phrase.

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, 3*Brazil, 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, 3*y, z	100	0	100	2	40
F	x, 3*y, 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, 3*y,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'
```

Notes

In an AND query, the score returned is the score of the lowest query term. In the example above, 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 a 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. 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.

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)
```

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.

Notes

Each hierarchy in a thesaurus represents a distinct, separate branch, corresponding to the four broader term operators. In a broader term query, Oracle only expands the query using the branch corresponding to the specified broader term operator.

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 11.

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'
```

Notes

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 the documents in the database.

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.

Syntax

Syntax	Description
?*term*	Expands a query to include all terms with similar spellings as the specified term.

Examples

Input	Expands To
?cat	cat cats calc case
?feline	feline defined filtering
?apply	apply apple applied April
?read	lead real

Notes

Fuzzy 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. Also, the Japanese lexer provides limited fuzzy matching.

In addition, if fuzzy returns a stopword, the stopword is not included in the query or highlighted by CTX_DOC.HIGHLIGHT or CTX_DOC.MARKUP.

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.

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 occurrences of *term2* from occurrences of *term1*.

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 number of occurrences of *cars* minus the number of occurrences of *Ford*. If a document does not contain any occurrences of the term *Ford*, no value is subtracted from the score.

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.

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.

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)'
```

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 11.

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 above, 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 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 the example above, 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'
```

Notes

In an OR query, the score returned is the score for the highest query term. In the example above, 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.

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 11.

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.

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 11.

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..
```

Notes

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.

The iMT 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

Notes

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 emp
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.

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 11.

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.

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.

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 11.

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
TR(*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.

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.

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 11.

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.

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 11.

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 4–1 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 4–1

	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 Double-Truncated Query Performance

When your wildcard queries are left- and double-truncated, you can improve query performance by creating a substring index. Substring indexes improve the query performance for all types of left-truncated wildcard searches such as *%ed*, *_ing*, or *%benz%*.

> **See Also:** For more information about creating substring indexes, see "BASIC_WORDLIST" in Chapter 3.

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'
```

The above 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

Scoring

The WITHIN operator has no effect on score.

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.

5

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 *inter*Media 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 interMedia 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
(none)	?	fuzzy
(none)	{ }	escape characters (multiple)
(none)	\	escape character (single)
(none)	( )	grouping characters
(none)	[ ]	grouping characters
MINUS	-	MINUS
NEAR	;	NEAR
NOT	~	NOT
NT	(none)	Narrower Term
NTG	(none)	Narrower Term Generic
NTI	(none)	Narrower Term Instance
NTP	(none)	Narrower Term Partitive
OR	|	OR
PT	(none)	Preferred Term
RT	(none)	Related Term
(none)	$	stem
(none)	!	soundex

Reserved Word	Reserved Character	Operator
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

6

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)
- `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_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)

> **See Also:** To learn more about the default values for these parameters, see "System Parameters" in Chapter 3.

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

"ctxsrv" in Chapter 11

7

CTX_DDL Package

This chapter provides reference information for using the CTX_DDL PL/SQL package to create and manage the objects 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_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_PREFERENCE	Creates a preference in the Text data dictionary
CREATE_SECTION_GROUP	Creates a section group in the Text data dictionary
CREATE_STOPLIST	Creates a stoplist.
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.

Name	Description
OPTIMIZE_INDEX	Optimize index.
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.

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.

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.

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 (:) 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:

```
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'
```

Notes

You can add attribute sections only to XML section groups.

When you use AUTO_SECTION_GROUP, attribute sections are created automatically. Attribute sections created automatically are named in the form tag@attribute.

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.

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.

tag

Specify the tag which marks the start of a section. For example, if the tag is <H1>, specify H1.

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.

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

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.add_zone_section('mygroup', 'author', 'meta@author');
end
```

After indexing with section group mygroup, you can query the document as follows:

```
'ken WITHIN author'
```

Notes

Oracle knows what the end tags look like from the group_type parameter you specify when you create the section group. The start tag you specify must be unique within a section group.

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.

You can define up to 64 field sections within a section group. Within the same group, section zone names and section field names cannot be the same.

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 4.

"Section Group Types" in Chapter 3.

CREATE_SECTION_GROUP

ADD_ZONE_SECTION

ADD_SPECIAL_SECTION

REMOVE_SECTION

DROP_SECTION_GROUP

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 as are zone and field sections. The start and end of special sections are detected when the Text index is created. Oracle supports two such sections: *paragraph* and *sentence*.

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;
```

Notes

The sentence and paragraph boundaries are determined by the lexer. Therefore, if the lexer cannot recognize the boundaries, no sentence or paragraph sections are indexed.

Related Topics

WITHIN operator in Chapter 4.

"Section Group Types" in Chapter 3.

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.

Example

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

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

Notes

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

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.

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;
```

Notes

Stop sections do not have section names and hence are not recorded in the section views.

Related Topics

ALTER INDEX in Chapter 2.

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.

Example

The following example adds the stoptheme `banking` to the stoplist `mystop`:

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

Notes

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

Related Topics

CREATE_STOPLIST

REMOVE_STOPTHEME

DROP_STOPLIST

ABOUT operator in Chapter 4.

ADD_STOPWORD

Adds 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
);
```

stoplist_name
Specify the name of the stoplist.

stopword
Specify the stopword to be added.

Example

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;
```

Note: You can add stopwords after you create the index with ALTER INDEX.

Notes

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

Related Topics

CREATE_STOPLIST

REMOVE_STOPWORD

DROP_STOPLIST

ALTER INDEX in Chapter 2.

Appendix E, "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.

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 NLS 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 NLS name of abbreviation of language.

Specify DEFAULT to assign a default sub-lexer 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 NLS 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');
```

Notes

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.

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.

tag
Specify the pattern which marks the start of a section. For example, if <H1> is the HTML tag, specify H1 for tag.

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.

Example

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

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

Oracle knows what the end tags look like from the group_type parameter you specify when you create the section group. The start tag you specify must be unique within a section group.

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.

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 4.

"Section Group Types" in Chapter 3.

CREATE_SECTION_GROUP

ADD_FIELD_SECTION

ADD_SPECIAL_SECTION

REMOVE_SECTION

DROP_SECTION_GROUP

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 object.

> **See Also:** For a complete list of preference objects and their associated attributes, see Chapter 3, "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 Objects" in Chapter 3.

Creating Master/Detail Relationship

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. 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 3.

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: For more information about storage, see "Storage Objects" in Chapter 3.

Creating Preferences with No Attributes

When you create preferences with objects 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 2.

ALTER INDEX in Chapter 2.

Chapter 3, "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	This is the default. Use this group type when you define no sections or when you define *only* SENTENCE or PARAGRAPH sections.
BASIC_SECTION_GROUP	Use this group type for defining sections where the start and end tags are of the form <A> and </A>.
HTML_SECTION_GROUP	Use this group type for defining sections in HTML documents.
XML_SECTION_GROUP	Use this group type for defining sections in XML-style tagged 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. 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.) ■ The length of the indexed tags including prefix and namespace cannot exceed 64 characters. Tags longer than this are not indexed.
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 4.

"Section Group Types" in Chapter 3.

ADD_ZONE_SECTION

ADD_FIELD_SECTION

ADD_SPECIAL_SECTION

REMOVE_SECTION

DROP_SECTION_GROUP

CREATE_STOPLIST

Creates a new, empty stoplist. Stoplists can contain words or themes that are not to be indexed.

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_name
Specify the name of the stoplist to be created.

Example

The following code creates a stoplist called `mystop`:

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

Notes

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

Related Topics

ADD_STOPWORD

ADD_STOPCLASS

ADD_STOPTHEME

DROP_STOPLIST

CREATE INDEX in Chapter 2.

ALTER INDEX in Chapter 2.

Appendix E, "Supplied Stoplists"

DROP_PREFERENCE

The DROP_PREFERENCE procedure deletes the specified preference from the Text data dictionary.

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;
```

Notes

Dropping a preference does not affect indexes that have been created using that preference.

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.

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;
```

Notes

When you drop a stoplist, you must recreate or rebuild the index for the change to take effect.

Related Topics

CREATE_STOPLIST

OPTIMIZE_INDEX

Optimizes the index. You can optimize on fast or full mode. This is the same as optimizing with ALTER INDEX.

Syntax

```
CTX_DDL.OPTIMIZE_INDEX(
  idx_name  in  varchar2,
  optlevel  in  varchar2,
  maxtime   in  number    default null
);
```

idx_name
Specify the name of the index.

optlevel
Specify optimization level as a string, either FAST or FULL.

You can also specify this parameter as one of the symbols: CTX_DDL.OPTLEVEL_FAST or CTX_DDL.OPTLEVEL_FAST.

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 (garbage collection.)

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.

When you specify 0 for maxtime, Oracle performs minimal optimization.

Example

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;
```

Related Topics

ALTER INDEX in Chapter 2.

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
);
```

stoplist_name
Specify the name of the stoplist.

stopword
Specify the stopword to be removed from stoplist_name.

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, 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 Objects" in Chapter 3.
>
> For more examples of using SET_ATTRIBUTE, see CREATE_PREFERENCE.

SYNC_INDEX

Synchronizes the index to process inserts, updates, and deletes. This is the same as synchronizing with ALTER INDEX.

Syntax

```
ctx_ddl.sync_index(idx_name  in  varchar2 default NULL);
```

idx_name
Specify the name of the index.

Example

```
begin
ctx_ddl.sync_index(myindex);
end;
```

Related Topics

ALTER INDEX in Chapter 2.

UNSET_ATTRIBUTE

Removes a set attribute from a preference.

Syntax

```
CTX_DDL.UNSET_ATTRIBUTE(preference_name varchar2,
                        attribute_name  varchar2);
```

preference_name
Specify the name of the preference.

attribute_name
Specify the name of the attribute.

Example

Enabling/Disabling Alternate Spelling

The following example shows how you can enable alternate spelling for German and disable alternate spelling with CTX_DDL.UNSET_ATTRIBUTE:

```
begin
ctx_ddl.create_preference('GERMAN_LEX', 'BASIC_LEXER');
ctx_ddl.set_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING', 'GERMAN');
end;
```

To disable alternate spelling, use the CTX_DDL.UNSET_ATTRIBUTE procedure as follows:

```
begin
ctx_ddl.unset_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING');
end;
```

Related Topics

SET_ATTRIBUTE

8

CTX_DOC Package

This chapter describes the CTX_DOC PL/SQL package for requesting document services. 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
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

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 CLOB,
        query_id    IN NUMBER DEFAULT 0,
        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.
- 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.

> **See Also:** For more information about the structure of the filter result table, see "Filter Table" in Appendix B.

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.

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, '0', 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;
```

Notes

Before CTX_DOC.FILTER is called, the result table specified in restab must exist.

When textkey is a composite textkey, you must encode the composite textkey string using CTX_DOC.PKENCODE.

When query_id is not specified or set to NULL, it defaults to 0. You must manually truncate the table specified in restab.

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 / theme summaries.

Syntax 1: In-Memory Storage

```
CTX_DOC.GIST(
                index_name    IN VARCHAR2,
                textkey       IN VARCHAR2,
                restab        IN OUT CLOB,
                query_id      IN NUMBER DEFAULT 0,
                glevel        IN VARCHAR2 DEFAULT 'P',
                pov           IN VARCHAR2 DEFAULT NULL,
                numParagraphs IN NUMBER DEFAULT 16,
                maxPercent    IN NUMBER DEFAULT 10);
```

Syntax 2: Result Table Storage

```
CTX_DOC.GIST(
                index_name    IN VARCHAR2,
                textkey       IN VARCHAR2,
                restab        IN VARCHAR2,
                query_id      IN NUMBER DEFAULT 0,
                glevel        IN VARCHAR2 DEFAULT 'P',
                pov           IN VARCHAR2 DEFAULT NULL,
                numParagraphs IN NUMBER DEFAULT 16,
                maxPercent    IN NUMBER DEFAULT 10);
```

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.

- 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:** For more information about the structure of the Gist result table, see "Gist Table" in Appendix B.

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 / 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 / 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 CTX_DOC.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/theme summaries. The default is 16.

Note: The numParagraphs parameter is used only when this parameter yields a smaller Gist/theme summary size than the Gist/theme summary size yielded by the maxPercent parameter.

This means that the system always returns the smallest size Gist/theme summary.

---

maxPercent

Specify the maximum number of document paragraphs (or sentences) selected for the document Gist/theme summaries as a percentage of the total paragraphs (or sentences) in the document. The default is 10.

Note: The maxPercent parameter is used only when this parameter yields a smaller Gist/theme summary size than the Gist/theme summary size yielded by the numParagraphs parameter.

This means that the system always returns the smallest size Gist/theme summary.

Examples

In-Memory Gist

The following example generates a non-default size generic Gist of at most ten paragraphs. The result is stored in memory in a CLOB locator. The code then de-allocates the returned CLOB locator after using it.

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

begin
 ctx_doc.gist('newsindex','34','gklob',1,glevel => 'P',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 The following example returns a default sized paragraph level Gist for document 34 as well as a theme summary for all the themes in the document:

```
begin
ctx_doc.gist('newsindex','34','CTX_GIST',1,glevel => 'P');
end;
```

The following example generates a non-default size Gist of at most ten paragraphs:

```
begin
ctx_doc.gist('newsindex','34','CTX_GIST',1,glevel => 'P',pov => 'GENERIC',
numParagraphs => 10);
end;
```

The following example generates a Gist whose number of paragraphs is at most ten percent of the total paragraphs in document:

```
begin
ctx_doc.gist('newsindex','34','CTX_GIST',1, glevel =>'P',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,glevel =>'P', pov => 'insects');
end;
```

Notes

By default when using result table storage and you specify no pov, this procedure generates up to 50 theme summaries for a document. As a result, CTX_DOC.GIST creates a maximum of 51 gists for each document: one theme summary for each theme and one Gist for the entire document.

When you use in-memory storage, CTX_DOC.GIST creates only one Gist.

When *textkey* is a composite textkey, you must encode the composite textkey string using the CTX_DOC.PKENCODE procedure as in the second example above.

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 DEFAULT NULL,
          restab      IN OUT HIGHLIGHT_TAB,
          query_id    IN NUMBER   DEFAULT 0,
          plaintext   IN BOOLEAN  DEFAULT FALSE);
```

Syntax 2:Result Table Storage

```
CTX_DOC.HIGHLIGHT(
          index_name  IN VARCHAR2,
          textkey     IN VARCHAR2,
          text_query  IN VARCHAR2 DEFAULT NULL,
          restab      IN VARCHAR2 DEFAULT NULL,
          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.
- 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.

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.

> **See Also:** For more information about the structure of the highlight result table, see "Highlight Table" in Appendix B.

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.

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.

Notes

Before CTX_DOC.HIGHLIGHT is called, the result table specified in *restab* must exist.

When *textkey* is a composite textkey, you must encode the composite textkey string using the CTX_DOC.PKENCODE procedure.

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.

When query_id is not specified or set to NULL, it defaults to 0. You must manually truncate the table specified in restab.

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 CLOB,
    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);
```

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.
- 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.

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.

> **See Also:** For more information about the structure of the markup result table, see "Markup Table" in Appendix B.

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.

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	
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.

```
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;
```

Notes

Before CTX_DOC.MARKUP is called, the result table specified in restab must exist.

When textkey is a composite textkey, you must encode the composite textkey string using the CTX_DOC.PKENCODE procedure.

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.

When query_id is not specified or set to NULL, it defaults to 0. You must manually truncate the table specified in restab.

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.

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 up to fifty themes for a document. Each theme is stored as a row in either a result table or 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 THEME_TAB,
               query_id        IN NUMBER DEFAULT 0,
               full_themes     IN BOOLEAN DEFAULT FALSE);
```

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);
```

index_name

Specify the name of the index for the column in which the document for the list of themes is stored.

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.
- 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 to a table specify the name of the table.

> **See Also:** For more information about the structure of the theme result table, see "Theme Table" in Appendix B.

To store results to 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.

Examples

In-Memory Themes

The following example generates the themes for document 1 and stores them in an in-memory table, declared as `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);
 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 themes where each element in the list is a single theme, issue:

```
begin
ctx_doc.themes('newsindex','34','CTX_THEMES',1,full_themes => FALSE);
end;
```

Full Themes

To obtain a list of themes where each element in the list is a hierarchical list of parent themes, issue:

```
begin
ctx_doc.themes('newsindex','34','CTX_THEMES',1,full_themes => TRUE);
end;
```

Notes

When textkey is a composite key, you must encode the composite textkey string using the CTX_DOC.PKENCODE procedure.

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
END_LOG	Halts logging of index and document services requests.
LOGFILENAME	Returns the name of the current log file.
START_LOG	Starts logging index and document service requests.

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;
```

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 use for generating query feedback, counting hits, and creating stored query expressions.

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 SQE 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 interMedia 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_word(
  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
);
```

Syntax 2: To Store Results in Memory

```
ctx_query.browse_word(
  index_name  in      varchar2,
  seed        in      varchar2,
  resarr      in out  browse_tab,
  numwords    in      number   default 10,
  direction   in      varchar2 default BROWSE_AROUND
);
```

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`. 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 length of the produced list in number of words. Specify a number between 1 and 1000.

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.

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

```
declare
  resarr ctx_query.browse_tab;
begin
ctx_query.browse_words('myindex','dog',resarr,5,CTX_QUERY.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
) 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.

Notes

Specifying FALSE returns a less accurate number but runs faster.

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.

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);
```

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.

explain_table
Specify the name of the table used to store representation of the parse tree for *text_query*.

See Also: For more information about the structure of the explain table, see "EXPLAIN Table" in Appendix B.

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.

Notes

You must have at least INSERT and DELETE privileges on the table used to store the results from EXPLAIN.

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.

You cannot use EXPLAIN with remote queries.

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 4, "Query Operators"

Appendix I, "Stopword Transformations"

HFEEDBACK

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.

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,
           );
```

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 B.

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.

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 B.

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 SQE to be removed.

Examples

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

STORE_SQE

This procedure creates a stored query expression (SQE). Only the query definition is stored.

Syntax

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

query_name
Specify the name of the SQE 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;
```

Notes

SQEs support all of the Text query expression operators. SQEs also support all of the special characters and other components that can be used in a query expression, including other SQEs.

Users are allowed to create and remove SQEs owned by them. Users are allowed to use SQEs owned by anyone. The CTXSYS user can create or remove SQEs for any user.

11

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 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 and returns the ID for the thesaurus.
DROP_PHRASE	Removes a phrase from thesaurus.
DROP_RELATION	Removes a relation between two phrases.
DROP_THESAURUS	Drops the specified thesaurus from the thesaurus tables.

Name	Description
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.

See Also: For more information about the thesaurus operators, see Chapter 4, "Query Operators".

ALTER_PHRASE

Alters an existing phrase in the thesaurus.

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_MAKE_PT	Make phrase the preferred term. Existing preferred terms in the synonym ring becomes non-preferred synonym.	(none)
SN	Change the scope note on the phrase.	Specify new scope note.

operand
Specify argument to the alter operation. See table for op.

NOTES

Only CTXSYS or thesaurus owner can alter a phrase.

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`.

Notes

Only thesaurus owner and ctxsys are allowed to invoke this function on a given thesaurus.

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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:

```
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:

```
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(lpad(' ', 2*xtab(i).xlevel) ||
    xtab(i).xrel || ' ' || xtab(i).xphrase);
  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 4

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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:

```
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 4

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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:

```
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 4

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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 4

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 thesarus.SYN (i.e. *phrase* is synonymous term for *relname*)

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

Example: 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);
```

thes_name
Specify the name of the thesaurus to be created.

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.

Examples

```
begin
   ctx_thes.create_thesaurus('tech_thes', FALSE);
end;
```

Notes

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.

To enter phrases in the thesaurus, use CTX_THES.CREATE_PHRASE or use the Thesaurus Maintenance screen in the System Administration tool.

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.

Notes

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;
```

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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}|{wild cat}|{Persian
cat}|{Siamese 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 4

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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 4

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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 4

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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 4

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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 4

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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 4

This function returns all synonyms of a phrase as recorded in the specified thesaurus.

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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 4

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.

Syntax

```
CTX_THES.THES_TT(restab IN OUT NOCOPY EXP_TAB,
                 tname  IN VARCHAR2 DEFAULT 'DEFAULT');
```

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

tname
Specify thesaurus name. If not specified, system default thesaurus is used.

Returns

This procedure returns top terms only to the table you specify with restab.

Notes

Since this function 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.

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 *inter*Media 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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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 4

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 *inter*Media 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: For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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 4

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:** For more information about EXP_TAB, see "CTX_THES Result Tables and Data Types" in Appendix B, "Result Tables".

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.rt('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 4

12

Executables

This chapter discusses the executables shipped with *inter*Media Text. The following topics are discussed:

- `ctxsrv`
- `ctxload`
- Knowledge Base Extension Compiler (`ctxkbtc`)

ctxsrv

You use the ctxsrv server daemon for background DML processing. You can start it from the command line or with the interMedia Text Manager administration tool.

This server synchronizes the index with ALTER INDEX at regular intervals.

Note: ctxsrv can *only* be executed by the Oracle user, CTXSYS.

Syntax

```
ctxsrv [-user ctxsys/passwd[@sqlnet_address]]
       [-personality M]
       [-logfile log_name]
       [-sqltrace]
```

-user
Specify the username and password for the Oracle user CTXSYS.

The username and password can be immediately followed 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 need not specify a value for sqlnet_address to connect to the database.

Note: If you do not specify -user in the ctxsrv command-line, you are prompted to enter the required information in the format: CTXSYS/password where password is the password for CTXSYS.

This is useful if you wish to mask the CTXSYS password from other users of the machine on which the server is running.

-personality
Specify the personality mask for the server started by ctxsrv. The only possible value is M and M is the default.

-logfile
Specify the name of a log file to which the server writes all session information and errors.

-sqltrace
Enables the server to write to a trace file in the directory specified by the USER_DUMP_DEST initialization parameter.

See Also: For more information about SQL trace and the USER_DUMP_DEST initialization parameter, see *Oracle8 Administrator's Guide*.

Examples

The following example starts a server and writes all server messages to a file named `ctx.log`:

```
ctxsrv -user ctxsys/ctxsys -personality M -log ctx.log &
```

The following example starts a server and writes all server messages to a file named `ctx.log`. Because `-user` is not specified, the server prompts you to enter a user:

```
ctxsrv -log ctx.log

...
Copyright (c) Oracle Corporation 1979, 1998.  All rights reserved.
...
Enter user:
```

At the prompt, enter `CTXSYS/password`, where `password` is the password assigned to the CTXSYS user.

Unix Users: In this example, the process is *not* run in the background.

In environments where you can run processes in the background, if you do not specify `-user` in the ctxsrv command-line, you must run the server process in the foreground or pass a value for `-user` to ctxsrv from an operating system file.

For example:

```
ctxsrv -log ctx.log < pword.txt
```

The file must contain a single line consisting of the following text: `CTXSYS/password`

If you pass a value to `ctxsrv` from a file, `ctxsrv` does not prompt you to enter a user.

Notes

Viewing Pending Updates

Pending index updates are stored in the DML queue. To view this queue, you can use the CTX_PENDING or CTX_USER_PENDING views.

You can also use the *inter*Media Text Manager administration tool, which is part of the Oracle Enterprise Manager.

Viewing DML Errors

You can view DML errors with the CTX_INDEX_ERRORS or CTX_USER_INDEX_ERRORS views.

Index Fragmentation

Background DML with `ctxsrv` scans for DML constantly by polling the DML queue. This leads to new additions being indexed automatically and quickly. However, background DML also tends to process documents in smaller batches, which increases index fragmentation.

However, when you synchronize the index manually with ALTER INDEX, the batches are usually larger and thus there is less index fragmentation.

Shutting Down the Server

You can shut down `ctxsrv` with

- CTX_ADM.SHUTDOWN.
- The *inter*Media Text Manager administration tool available with Oracle Enterprise Manager

Related Topics

ALTER INDEX

CTX_ADM.SHUTDOWN in Chapter 6.

The following views in Appendix H, "Views":

- CTX_PENDING
- CTX_USER_PENDING
- CTX_INDEX_ERRORS

- CTX_USER_INDEX_ERRORS

The *inter*Media Text Manager

For more information on starting servers with the administration tool, see the online help for the *inter*Media Text Manager. This administration tool is a Java application integrated with the Oracle Enterprise Manager.

ctxload

You use ctxload to perform the following operations:

- Thesaurus Importing and Exporting
- Text Loading
- Document Updating/Exporting

Thesaurus Importing and Exporting

Use ctxload to load a thesaurus from an import file into the iMT thesaurus tables.

An import file is an ASCII flat file that contains entries for synonyms, broader terms, narrower terms, or related terms which can be used to expand queries.

ctxload can also be used to export a thesaurus by dumping the contents of the thesaurus into a user-specified operating-system file.

> **See Also:** For examples of import files for thesaurus importing, see "Structure of ctxload Thesaurus Import File" in Appendix D.

Text Loading

You can use ctxload to load text from a load file into a LONG or LONG RAW column in a table.

> **Suggestion:** If the target table does not contain a LONG or LONG RAW column or you do not want to load text into a LONG or LONG RAW column, you can use SQL*Loader to populate the table with text.
>
> For more information on loading with SQL*Loader, see "SQL*Loader Example" in Appendix D.

A load file is an ASCII flat file that contains the plain text, as well as any structured data (title, author, date, etc.), for documents to be stored in a text table; however, in place of the text for each document, the load file can store a pointer to a separate file that holds the actual text (formatted or plain) of the document.

Note: The ctxload utility does not support load files that contain both embedded text and file pointers. You must use one method or the other when creating a load file.

The ctxload utility creates one row in the table for each document identified by a header in the load file.

See Also: For examples of load files for text loading, see "Structure of `ctxload` Text Load File" in Appendix D.

Document Updating/Exporting

The `ctxload` utility supports updating database columns from operating system files and exporting database columns to files, specifically LONG RAW, LONG, BLOB and CLOB columns.

Note: The updating/exporting of data is performed in sections to avoid the necessity of a large amount of memory (up to 2 Gigabytes) for the update/fetch buffer.

As a result, a minimum of 16 Kilobytes of memory is required for document update/export.

ctxload Syntax

```
ctxload -user username[/password][@sqlnet_address]
        -name object_name
        -file file_name
       [-pk primary_key]
       [-export]
       [-update]
       [-thes]
       [-thescase y|n]
       [-thesdump]
       [-separate]
       [-longsize n]
       [-date date_mask]
       [-log file_name]
       [-trace]
       [-commitafter n]
```

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.

When `ctxload` is used to update a single row in a text column, use file_name to specify the file that stores the text to be inserted into the text column. You identify the destination row with -pk.

When `ctxload` is used to export a single row in a text column, use file_name to specify the file to which the text is exported. You identify the source row with -pk.

See Also: For more information about the structure of ctxload import files, see Appendix D, "Loading Examples".

Optional Arguments

-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 single cell in a database table into the operating system file specified by -file. `ctxload` exports the LONG, LONG RAW, 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 single cell in a database table with the contents of the operating system file specified by -file. ctxload updates the LONG, LONG RAW, CLOB or BLOB column in for the row specified by -pk.

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

-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.

-separate
For text loading, include this parameter to specify that the text of each document in the load file is a pointer to a separate text file. This instructs `ctxload` to load the contents of each text file in the LONG or LONG RAW column for the specified row.

-longsize n
For text loading, specify the maximum number of kilobytes to load into the LONG or LONG RAW column.

The minimum value is 1 (that is 1 Kb) and the maximum value is machine dependent.

Note: You must enter the value for longsize as a number only. Do not include a *K* or *k* to indicate kilobytes.

-date
Specify the TO_CHAR date format for any date columns loaded using ctxload.

> **See Also:** For more information about the available date format models, see *Oracle8i SQL Reference*.

-log
Specify the name of the log file to which `ctxload` writes any national-language supported (NLS) 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 *Oracle8 Administrator's Guide*.

-commitafter n
Specify the number of rows (documents) that are inserted into the table before a commit is issued to the database. The default is 1.

Examples

This section provides examples for some of the operations that ctxload can perform.

> **See Also:** For more document loading examples, see Appendix D, "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
```

Exporting a Single Text Field

The following example exports a single text field identified by the primary key value of 1 to the file myfile. The index myindex identifies the text column.

```
ctxload -user scott/tiger -export -name myindex -file myfile -pk 1
```

To export a single text field identified by a compound primary key, you must enclose the primary keys with quotes and separate the values with commas as follows:

```
ctxload -user scott/tiger -export -name myindex -file myfile -pk "Oracle,1"
```

Updating a Single Text Field

The following example updates a single text field identified by primary key value of 1 with the contents of myfile. The index myindex identifies the text column.

```
ctxload -user scott/tiger -update -name myindex -file myfile -pk 1
```

To update a single text field identified by a compound primary key, you must enclose the primary key with quotes and separate the values with commas as follows:

```
ctxload -user scott/tiger -update -name myindex -file myfile -pk "Oracle,1"
```

Knowledge Base Extension Compiler (ctxkbtc)

The ctxkbtc compiler takes one or more specified thesauri and compiles them with the *inter*Media Text knowledge base to create an extended knowledge base. The extended information can be application-specific terms and relationships.

The extended knowledge base overrides any terms and relationships in the knowledge base where there is overlap. The extended knowledge base is accessed during tasks that use the knowledge base, such as theme indexing, processing ABOUT queries in English, and extracting document themes with document services.

> **See Also:** For more information about the knowledge base packaged with *inter*Media Text, see Appendix J, "Knowledge Base - Category Hierarchy".
>
> For more information about the ABOUT operator, see ABOUT operator in Chapter 4.
>
> For more information about document services, see Chapter 8, "CTX_DOC Package".

Syntax

```
ctxkbtc -user uname/passwd
        [-name thesname1 [thesname2 ... thesname16]]
        [-revert]
        [-verbose]
        [-log filename]
```

-user
Specify the username and password for the administrator creating an extended knowledge base.

-name
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.

-revert
Reverts the extended knowledge base to the default knowledge base provided by *inter*Media Text.

-verbose
Displays all warnings and messages, including non-NLS 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.

Usage Notes

Knowledge base extension cannot be performed when theme indexing is being performed.

In addition, any SQL sessions that are using *inter*Media 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.

Running ctxkbtc twice removes the previous extension.

Before being compiled, each thesaurus must be loaded into *inter*Media Text case sensitive with the "-thescase Y" option in ctxload.

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.

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.

Linking New Terms to Existing Terms

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 J, "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

You purchase a medical thesaurus `medthes` containing a 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
```

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
```

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

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

Description of Parameter	Limit
Number of separate thesauri that can be compiled into a user extension to the KB	16

A

Query Tuning

This appendix discusses how to tune your queries for better response time. The following topics are covered:

- Optimizing Queries with Statistics
- Optimizing Queries for Response Time
- Optimizing Queries for Throughput
- Tuning Queries with Blocking Operations

Optimizing Queries with Statistics

Query optimization with statistics uses the collected statistics on the tables and indexes in a query to select an execution plan that can process the query in the most efficient manner. The optimizer attempts to choose the best execution plan based on the following parameters:

- the selectivity on the CONTAINS predicate
- the selectivity of other predicates in the query
- the CPU and I/O costs of processing the CONTAINS predicates

The following sections describe how to use statistics with the extensible query optimizer. Optimizing with statistics allows for a more accurate estimation of the selectivity and costs of the CONTAINS predicate and thus a better execution plan.

Collecting Statistics

By default, the extensible query optimizer is enabled. To use the extensible optimizer, you must calculate the statistics on the table you query. To do so, issue the following statement:

```
ANALYZE TABLE <table_name> COMPUTE STATISTICS;
```

Alternatively, you can estimate the statistics on a sample of the table as follows:

```
ANALYZE TABLE <table_name> ESTIMATE STATISTICS 1000 ROWS;
```

or

```
ANALYZE TABLE <table_name> ESTIMATE STATISTICS 50 PERCENT;
```

These statement collects statistics on all the objects associated with table_name including the table columns and any indexes (b-tree, bitmap or Text domain) associated with the table. You can issue the above ANALYZE command as many times as necessary to re-collect the statistics on a table.

> **See Also:** For more information on the ANALYZE command, see *Oracle8i SQL Reference* and *Oracle8i Designing and Tuning for Performance*.

By collecting statistics on the Text domain index, the extensible query optimizer is able to do the following:

- estimate the selectivity of the CONTAINS predicate

- estimate the I/O and CPU costs of using the Text index, that is, the cost of processing the CONTAINS using the domain index
- estimate the I/O and CPU costs of each invocation of CONTAINS() function

Knowing the selectivity of a CONTAINS predicate is useful for queries that contain more than one predicate, such as in structured queries. This way the extensible query optimizer can better decide whether to use the domain index to evaluate CONTAINS or to apply the CONTAINS predicate as a post filter.

Example

Consider the following structured query:

```
select score(1) from tab where contains(txt, 'freedom', 1) > 0 and author =
'King' and year > 1960;
```

Assume the `author` column is of type VARCHAR2 and the `year` column is of type NUMBER. Assume that there is a b-tree index on the `author` column.

Also assume that the structured `author` predicate is highly selective with respect to the CONTAINS predicate and the year predicate; that is, the structured predicate (`author = 'King'`) returns a much smaller number of rows with respect to the `year` and CONTAINS predicates individually, say 5 rows versus 1000 and 1500 rows respectively.

In this situation, Oracle can execute this query more efficiently by first doing a b-tree index range scan on the structured predicate (`author = 'King'`), followed by a table access by rowid, and then applying the other two predicates to the rows returned from the b-tree table access.

Without associating a statistics type with indextype `context`, the extensible query optimizer will always choose to process the CONTAINS() predicate using the text domain index.

Note: When the statistics are not collected for a Text index, the behavior is the same as not enabling the extensible query optimizer.

Re-Collecting Statistics

You can re-collect statistics on a single index by issuing any of the following statements:

```
ANALYZE INDEX <index_name> COMPUTE STATISTICS;
```

```
or

ANALYZE INDEX <index_name> ESTIMATE STATISTICS SAMPLE 1000 ROWS;

or

ANALYZE INDEX <index_name> ESTIMATE STATISTICS SAMPLE 50 PERCENT;
```

Deleting Statistics

You can delete the statistics associated with a table by issuing:

```
ANALYZE TABLE <table_name> DELETE STATISTICS;
```

You can delete statistics on one index by issuing the following statement:

```
ANALYZE INDEX <index_name> DELETE STATISTICS;
```

Disabling and Enabling the Extensible Query Optimizer

By default the extensible query optimizer is enabled. To disable the extensible query optimizer, issue the following statements:

```
DISASSOCIATE STATISTICS FROM INDEXTYPES ConText;
DISASSOCIATE STATISTICS FROM PACKAGES ctx_contains;
```

After disabling the extensible query optimizer, you can re-enable it. To do so, issue the following SQL statements as CTXSYS:

```
ASSOCIATE STATISTICS WITH INDEXTYPES ConText USING textoptstats;
ASSOCIATE STATISTICS WITH PACKAGES ctx_contains USING textoptstats;
```

Optimizing Queries for Response Time

By default, Oracle optimizes queries for throughput. This results in queries returning all rows in shortest time possible.

However, in many cases, especially in a web-application scenario, queries must be optimized for response time, when you are only interested in obtaining the first n hits of a potentially large hitlist in the shortest time possible.

The following sections describe how to optimize Text queries for response time. You can do so in two ways:

- using FIRST_ROWS hint
- using CHOOSE and DOMAIN_INDEX_SORT hints

> **Note:** Although both methods optimize for response time, the execution plans of the two methods obtained with EXPLAIN PLAN might be different for a given query.

Better Response Time with FIRST_ROWS

You can change the default query optimizer mode to optimize for response time using the FIRST_ROWS hint. When queries are optimized for response time, Oracle returns the first n rows in the shortest time possible.

For example, consider the following PL/SQL block that uses a cursor to retrieve the first 20 hits of a query and uses the FIRST_ROWS hint to optimize the response time:

```
declare
cursor c is
select /*+ FIRST_ROWS */ pk, score(1), col from ctx_tab
           where contains(txt_col, 'test', 1) > 0 order by score(1) desc;
begin
for i in c
loop
insert into t_s values(i.pk, i.col);
exit when c%rowcount > 21;
end loop;
end;
/
```

The cursor c is a SELECT statement that returns the rowids that contain the word *test* in sorted order. The code loops through the cursor to extract the first 20 rows. These rows are stored in the temporary table `t_s`.

With the FIRST_ROWS hint, Oracle instructs the Text index to return rowids in score-sorted order, if possible.

Without the hint, Oracle sorts the rowids after the Text index has returned *all* the rows in unsorted order that satisfy the CONTAINS predicate. Retrieving the entire result set as such takes time.

Since only the first 20 hits are needed in this query, using the hint results in better performance.

Note: Use the FIRST_ROWS hint when you need only the first few hits of a query. When you need the entire result set, do not use this hint as it might result in poorer performance.

In addition, the FIRST_ROWS hint can be used with or without enabling the extensible query optimizer.

Other Behavior with FIRST_ROWS

Besides instructing the Text index to return hits in score-sorted order, the FIRST_ROWS hint also tries to avoid blocking operations when optimizing queries for response time. Blocking operations include merge joins, hash joins and bitmap operations.

As a result, using the FIRST_ROWS hint to optimize for response time might result in a different execution plan than using CHOOSE with DOMAIN_INDEX_SORT, which also optimizes for response time.

You can examine query execution plans using the EXPLAIN PLAN command in SQL.

> **See Also:** For more information about the query optimizer and using hints such as FIRST_ROWS and CHOOSE, see *Oracle8i Concepts* and *Oracle8i Designing and Tuning for Performance*.
>
> For more information about the EXPLAIN PLAN command, see *Oracle8i SQL Reference*

Better Response Time with CHOOSE

When you use the CHOOSE or ALL_ROWS optimizer hints, the query is optimized for throughput. This is the default optimizer mode. In this mode, Oracle does not instruct the Text domain index to return score-sorted rows, choosing instead to sort all the rows fetched from the Text index.

To optimize for fast response time under CHOOSE or ALL_ROWS modes, you can use the DOMAIN_INDEX_SORT hint as follows:

```
declare
cursor c is
select /*+ CHOOSE DOMAIN_INDEX_SORT */ pk, score(1), col from ctx_tab
           where contains(txt_col, 'test', 1) > 0 order by score(1) desc;
begin
for i in c
loop
insert into t_s values(i.pk, i.col);
exit when c%rowcount > 21;
end loop;
end;
/
```

Note: Although you can optimize fro response with this method as well as with FIRST _ROWS by itself, the actual execution plans of the two methods obtained with EXPLAIN PLAN might be different for a given query.

See Also: For more information about the query optimizer and using hints such as FIRST_ROWS and CHOOSE, see *Oracle8i Concepts* and *Oracle8i Designing and Tuning for Performance*.

For more information about the EXPLAIN PLAN command, see *Oracle8i SQL Reference*

Optimizing Queries for Throughput

CHOOSE and ALL ROWS Modes

By default, queries are optimized for throughput under the CHOOSE and ALL_ROWS modes. When queries are optimized for throughput, Oracle returns *all* rows in the shortest time possible.

FIRST_ROWS Mode

In FIRST_ROWS mode, the extensible query optimizer optimizes for fast response time by having the Text domain index return score-sorted rows, if possible. This is the default behavior when you use the FIRST_ROWS hint.

If you want to optimize for better throughput under FIRST_ROWS, you can use the DOMAIN_INDEX_NO_SORT hint. Better throughput means you are interested in getting all the rows to a query in the shortest time.

The following example achieves better throughput by not using the Text domain index to return score-sorted rows. Instead, Oracle sorts the rows after all the rows that satisfy the CONTAINS predicate are retrieved from the index:

```
select /*+ FIRST_ROWS DOMAIN_INDEX_NO_SORT */ pk, score(1), col from ctx_tab
           where contains(txt_col, 'test', 1) > 0 order by score(1) desc;
```

See Also: For more information about the query optimizer and using hints such as FIRST_ROWS and CHOOSE, see *Oracle8i Concepts* and *Oracle8i Designing and Tuning for Performance*.

Tuning Queries with Blocking Operations

Issuing a query with more than one predicate can cause a blocking operation in the execution plan. For example, consider the following mixed query:

```
select docid from myindex where contains(text, 'oracle', 1) > 0
  AND colA > 5
  AND colB > 1
  AND colC > 3;
```

Assume that all predicates are unselective and colA, colB, and colC have bitmap indexes. The Oracle cost-based optimizer chooses the following execution plan:

```
TABLE ACCESS BY ROWIDS
  BITMAP CONVERSION TO ROWIDS
    BITMAP AND
      BITMAP INDEX COLA_BMX
      BITMAP INDEX COLB_BMX
      BITMAP INDEX COLC_BMX
      BITMAP CONVERSION FROM ROWIDS
        SORT ORDER BY
          DOMAIN INDEX MYINDEX
```

Since the BITMAP AND is a blocking operation, Oracle must temporarily save the rowid and score pairs returned from the interMedia Text domain index before executing the BITMAP AND operation.

Oracle attempts to save these rowid and score pairs in memory. However, when the size of the result set containing these rowid and score pairs exceeds the SORT_AREA_SIZE initialization parameter, Oracle spills these results to temporary segments on disk.

Since saving results to disk causes extra overhead, you can improve performance by increasing the SORT_AREA_SIZE parameter using ALTER SESSION as follows:

```
alter session set SORT_AREA_SIZE = <new memory size in bytes>;
```

For example, to set the buffer to approximately 8 megabytes, you can issue:

```
alter session set SORT_AREA_SIZE = 8300000;
```

> **See Also:** For more information on SORT_AREA_SIZE, see Oracle8i Reference.

B

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 B–1 describes the structure of the table to which CTX_QUERY.EXPLAIN writes its results.

Table B–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 B–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 B–3 for possible values.
OBJECT_NAME	VARCHAR2(80)	Section name, wildcard term, weight, or threshold value or term to lookup in the index.

Table B–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 B–2 shows the possible values for the OPERATION column of the explain and hfeedback tables.

Table B–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 B–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 B–4 describes the table to which CTX_QUERY.HFEEDBACK writes its results.

Table B–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 B–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 B–5 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 B–6.
PT_FEEDBACK	CTX_FEEDBACK_TYPE	Stores related feedback terms. See Table B–6.
NT_FEEDBACK	CTX_FEEDBACK_TYPE	Stores narrower feedback terms. See Table B–6.

OPTIONS Column Values

The following table list the values for the OPTIONS column of the feedback table.

Table B–5

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 B–6

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.)
rank	method	(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_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 B–7

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 B–8

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 document 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 B–9

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 B–10

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 B–11

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.

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 optionally 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.

C

Supported Filter 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 Inso Filtering Technology
- Supported Document Formats
- Unsupported Formats

About Inso Filtering Technology

Oracle8i *inter*Media Text uses document filtering technology licensed from Inso Corporation. 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 INSO 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 Inso Corporation and Adobe Corporation.

The following sections discuss the supported platforms and how to enable Inso filtering on the different platforms.

Supported Platforms

Inso filter technology is supported on the following platforms:

- Sun Solaris Sparc (2.4 - 2.6)
- IBM AIX RS 6000 (4.1.4 - 4.3)
- HP / UX HP9000 (9.0 - 11.0)
- Digital UNIX (4.0 and above)
- SGI IRIX (6.3 "New 32 -bit")
- NT on Intel (x86 NT 3.51 and above)
- DEC Alpha NT (4.0 and above)

Environment Variable Locations

All environment variables related to Inso filtering must made visible to *inter*Media Text. Set these variables in the following locations:

- listener.ora file. This makes the environment variables visible to the extproc PL/SQL process.
- The operating system shell from where `ctxsrv` server is started. This makes the environment variables visible to the `ctxsrv` process, which does background DML.

Considerations for UNIX Platforms

The following considerations apply to Solaris, IBM AIX, HP/UX, Digital UNIX, and SGI platforms:

- Ensure the *.flt files have execute permission granted to the operating system user running the Oracle database and ctxsrv server.
- Set the `$PATH` variable to the location of the *.flt files, in particular to the location of the file `isunx2.flt`.
- Set the `$HOME` environment variable to allow Inso technology to write files to a sub-directory (.inso) in `$HOME` directory.
- Access to a running X-Windows server is required to perform graphics conversion.

Solaris

Set the system's shared library path (`$LD_LIBRARY_PATH`) environment variable as well as `$PATH` environment variable to include `$ORACLE_HOME/ctx/lib`, which is the location of the shared libraries for Inso filtering.

IBM AIX

Set the system's shared library path (`$LIBPATH`) environment variable as well as `$PATH` environment variable to include `$ORACLE_HOME/ctx/lib`, which is the location of the shared libraries for Inso filtering.

You must include `/usr/lib:/lib` in `$LIBPATH`, because these directories are not searched by default once `$LIBPATH` environment variable is set.

HP/UX

Set the system's shared library path (`$SHLIB_PATH`) environment variable as well as $PATH environment variable to include `$ORACLE_HOME/ctx/lib`, which is the location of the shared libraries for Inso filtering.

SGI

Set the system's shared library path (`$LD_LIBRARY_PATH`) environment variable as well as the `$PATH` environment variable to include `$ORACLE_HOME/ctx/lib`, which is the location of the shared libraries for Inso filtering.

Digital UNIX

Set the system's shared library path (`$LD_LIBRARY_PATH`) environment variable as well as the $PATH environment variable to include `$ORACLE_HOME/ctx/lib`, which is the location of the shared libraries for Inso filtering.

Filtering Vector Graphic Formats

Follow these steps to filter vector graphic formats on UNIX platforms:

- Run an X server to filter vector graphic formats (but not bitmap file formats). If no X server exists (system detects no X libraries), no vector graphic conversion can be 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` will tell 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 *inter*Media 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 Inso Corporation's HTML export technology and is licensed from Inso Corporation.

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	All versions
IBM Revisable Form Text	All versions
IBM FFT	Versions through 3.0
Microsoft Rich Text Format (RTF)	All versions

Word Processing - DOS

Format	Version
DEC WPS Plus	Versions through 4.1
DisplayWrite 2 & 3 (TXT)	All versions
DisplayWrite 4 & 5	Versions through Release 2.0
Enable	Versions 3.0, 4.0 and 4.5
First Choice	Versions through 3.0

Format	Version
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 7.0
WordStar	Versions through 7.0
WordStar 2000	Versions through 3.0
XyWrite	Versions through III Plus

Word Processing - International

Format	Version
Ichitaro	Version 8

Word Processing - Windows

Format	Version
AMI / AMI Professional	Versions through 3.1
Corel WordPerfect for Windows	Versions through 8.0
JustWrite	Versions through 3.0
Legacy	Versions through 1.1
Lotus WordPro (NT on Intel only)	WordPro 96, 97 and SmartSuite for the Millennium
Microsoft Windows Works	Versions through 4.0
Microsoft Windows Write	Versions through 3.0
Microsoft Word 97	Word 97
Microsoft Word 2000	Beta 2
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
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 (Mac)	Versions through 2.0
MacWrite II	Version 1.1

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 97	SmartSuite 97
Lotus 1-2-3 for SmartSuite for the Millennium	SmartSuite for the Millennium
Lotus 1-2-3 Charts (DOS & Windows)	Versions through 5.0
Lotus 1-2-3 (OS/2)	Versions through 2.0
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	Beta 2
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

Format	Version
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
QuattroPro for Windows	Versions through 8.0
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
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

Format	Version
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	Versions 1.0, 1.1, and 1.2 Not supported on DEC Alpha NT. Japanese PDF not supported.

Presentation Formats

Format	Version
Corel Presentations	Version 8.0
Novell Presentations	Versions 3.0 and 7.0
Harvard Graphics for DOS	Versions 2.x & 3.x
Freelance 96 for Windows 95	Freelance 96
Freelance for Windows 95	SmartSuite 97 and Millennium
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	Beta 2
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 any textual information from graphics.

Format	Version
Binary Group 3 Fax	All versions
BMP (including RLE, ICO, CUR & OS / 2 DIB)	Windows
CDR (if TIFF image is embedded in it)	Corel Draw versions 2.0 - 5.0
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 13
EMF (NT on Intel Only)	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
JPEG	All versions
PCX	PC Paintbrush
PBM - Portable Bitmap	No specific version
PCD - Kodak Photo CD	Version 1
Perfect Works (Draw)	Novell version 2.0

Format	Version
PGM - Portable Graymap	No specific version
PIC	Lotus
PICT1 & PICT2 (Raster)	Macintosh Standard
PNG - Portable Network Graphics Internet Format	Version 1.0
PNTG	MacPaint
PPM - Portable Pixmap	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
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
vCard Electronic Business Card	No specific version

Unsupported Formats

Password protected documents and documents with password protected content are not supported by the Inso filter.

D

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
- Structure of ctxload Text Load 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 required constraint for a Text table. 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 Oracle8i 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, which is required in a Text table. 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 `loader1.dat` into the `articles_formatted` table. Each line in `loader1.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 *inter*Media 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 *inter*Media 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 (e.g. 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 (e.g. *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
```

Structure of ctxload Text Load File

The ctxload text load file must use the following format for each document, as well as any structured data associated with the document:

```
<TEXTSTART: col_name1=doc_data, col_name2=doc_data,...col_nameN=doc_data>
text. . .
<TEXTEND>
```

where:

<TEXTSTART: ... >
is a header marker that indicates the beginning of a document. It also may contain one or more of the following fields used to specify structured data for a document:

col_name
is the name of a column that will store structured data for the document.

doc_data
is the structured data that will be stored in the column specified in col_name.

text
is the text of the document to be loaded or the name (and location, if necessary) of an operating system file containing the text to be loaded.

Note: The data in *text* (either a string of text or a file name pointer) is treated by ctxload as literal data, including any non-alphanumeric characters or blank spaces that may occur. As a result, you must ensure that *text* exactly represents the data you wish ctxload to process.

For example, if you use ctxload to load text from separate files, the file names in the load file must exactly represent the name(s) of the operating-system file(s) containing the text. If any blank spaces are included in a file name, ctxload cannot locate the file and the text is not loaded.

<TEXTEND>
indicates the end of the document.

Load File Structure

The following conditions apply to the structure of the load file:

- for each document to be loaded, either the text of the document or a pointer to a separate file must be in the load file.
- embedded text and separate file pointers cannot be used together in the same load file
- if the text for your documents is embedded in the load file, the text must be in ASCII format
- if pointers to separate files are used, the text in the files can be in plain (ASCII) format or a proprietary format (e.g. MS Word)
- if the text in a separate file is in a proprietary format, the format must be supported by ConText and it must be loaded into a LONG RAW column
- each separate file must contain a single document (the contents of a separate file are stored as a single row in the table)

Load File Syntax

The following conditions apply to the syntax utilized in the text load file:

- <TEXTSTART:...> and <TEXTEND> *must* each start on a new line
- the structured data parameters within the <TEXTSTART:...> string do not have to be in any particular order
- a newline character (either hard or soft return) cannot occur between a col_name and the beginning of its associated doc_data

> **Note:** The entire value for doc_data does not have to be on the same line as the col_name; only the beginning of the value and the col_name must share the same line.

- the first col_name should be on the same line as the 'TEXTSTART:'
- the '>' character which indicates the end of the <TEXTSTART:...> string must be on the same line as the last doc_data field for the document
- structured and LONG data may span more than one line

- single quote-marks must be escaped in doc_data (e.g. *don't* must be entered as *don"t*)
- each <TEXTSTART: ... > string *must* be followed by the text of a document or a pointer to a separate file
- the text or file pointer must be placed after the complete <TEXTSTART: ... > string and must start on a new line
- the last character in the load file must be a newline character

Example of Embedded Text in Load File

The following example illustrates a correctly formatted text load file containing structured employee information, such as employee number (1000, 1024) and name (Joe Smith, Mary Jones), and the text for each document:

```
<TEXTSTART: EMPNO=1000, ELNAME='Smith', EFNAME='Joe'>
Joe has an interesting resume, includes...cliff-diving.
<TEXTEND>
<TEXTSTART: EMPNO=1024, EFNAME='Mary', ELNAME='Jones'>
Mary has many excellent skills, including...technical,
marketing, and organizational.  Team player.
<TEXTEND>
```

Example of File Name Pointers in Load File

The following example illustrates a correctly formatted text load file containing structured employee information, such as employee number (1000, 1024) and name (Joe Smith, Mary Jones), and a file name pointer for each document.

```
<TEXTSTART: EMPNO=1024, EFNAME='Mary', ELNAME='Jones'>
mjones.doc
<TEXTEND>
<TEXTSTART: EMPNO=1000, EFNAME='Joe', EFNAME='Smith'>
jsmith.doc
<TEXTEND>
```

Note: To use the load file in this example, you would have to specify the -separate argument when executing ctxload.

E

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
- Danish (DA)
- Dutch (NL)
- Finnish (FI)
- French (FR)
- German (DE)
- Italian (IT)
- Portuguese (PR)
- Spanish (ES)
- Swedish (SE)

English

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	

Danish (DA)

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)

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 (FI)

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 (FR)

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 (DE)

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 (IT)

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 (PR)

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 (ES)

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 (SE)

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	

F

Alternate Spelling Conventions

This appendix describes the alternate spelling conventions that *inter*Media 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 *inter*Media 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

The German alphabet is the English alphabet plus the additional characters: ä ö ü ß. The following table lists the alternate spelling conventions *inter*Media Text uses for these characters.

Character	Alternate Spelling Substitution
ä	ae
ü	ue
ö	oe
Ä	AE
Ü	UE
Ö	OE
ß	ss

Danish

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 *inter*Media Text uses for these characters.

Character	Alternate Spelling Substitution
æ	ae
ø	oe
å	aa
Æ	AE
Ø	OE
Å	AA

Swedish

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 *inter*Media Text uses for these characters.

Character	Alternate Spelling Substitution
ä	ae
å	aa
ö	oe
Ä	AE
Å	AA
Ö	OE

G

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 either with ALTER INDEX or by running `ctxsrv` in the background.

See Also: For more information about ALTER INDEX, see ALTER INDEX in Chapter 2.

For more information about `ctxsrv`, see "ctxsrv" in Chapter 11.

H

Views

This appendix lists all of the views provided by *inter*Media Text.

Overview

The system provides the following views:

- CTX_CLASSES
- CTX_INDEXES
- CTX_INDEX_ERRORS
- CTX_INDEX_OBJECTS
- 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_USER_INDEXES
- CTX_USER_INDEX_ERRORS
- CTX_USER_INDEX_OBJECTS
- 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_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, either INDEXED or INDEXING.
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_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 CTXSYS.

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.

CTX_OBJECT_ATTRIBUTE_LOV

This view displays the allowed values for certain object attributes provided by *inter*Media 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 ■ default_index_memory ■ default_datastore ■ default_filter_binary ■ default_filter_text ■ default_filter_file ■ default_section_html ■ default_section_text ■ default_lexer ■ default_wordlist ■ default_stoplist ■ default_storage ■ log_directory
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_ROWID	ROWID	ROWID to be indexed
PND_TIMESTAMP	DATE	Time of modification

CTX_PREFERENCES

This view displays preferences created by *inter*Media Text users, as well as all the system-defined preferences included with *inter*Media 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, or SPECIAL.
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

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.
SPL_TYPE	VARCHAR2(10)	Stop type, either STOP_WORD, STOP_CLASS, STOP_THEME.
SPW_WORD	VARCHAR2(80)	Stopword.

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 all *inter*Media Text users.

Column Name	Type	Description
THS_OWNER	VARCHAR2(30)	Thesaurus owner
THS_NAME	VARCHAR2(30)	Thesaurus name

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_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 DML errors 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_VALUES

This view displays attribute values for each object used in indexes. 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_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, or SPECIAL.
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. It is queryable by all users.

Column Name	Type	Description
SPL_NAME	VARCHAR2(30)	Name of stoplist.
SPL_COUNT	NUMBER	Number of stopwords

CTX_USER_STOPWORDS

This view displays stopwords in each stoplist. 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.

CTX_USER_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_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

I

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

J

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] water birds
[8] ducks, geese, and swans
[8] gulls and terns
[8] pelicans
[7] woodpeckers
[7] wrens
[6] fish
[7] boneless fish
[8] rays and skates
[8] sharks
[7] bony fish
[8] deep sea fish
[8] eels
[8] tropical fish
[7] jawless fish
[6] mammals
[7] anteaters and sloths
[8] aardvarks
[7] carnivores
[8] canines
[8] felines
[7] chiropterans
[7] elephants
[7] hoofed mammals
[8] cattle
[8] goats
[8] horses
[8] pigs
[8] sheep
[7] hyraxes
[7] marine mammals
[8] seals and walruses
[9] manatees
[8] whales and porpoises
[7] marsupials
[7] monotremes
[7] primates
[8] lemurs
[7] rabbits
[7] rodents
[6] reptiles
[7] crocodilians
[7] lizards
[7] snakes
[7] turtles
[3] biotechnology
[4] antibody technology
[5] immunoassays
[4] biometrics
[5] voice recognition technology
[4] genetic engineering
[4] pharmaceutical industry
[5] anesthetics
[6] general anesthetics
[6] local anesthetics
[5] antagonists and antidotes
[5] antibiotics, antimicrobials, and antiparasitics
[6] anthelmintics
[6] antibacterials
[7] antimalarials
[7] antituberculars and antileprotics
[6] antifungals
[6] antivirals
[6] local anti-infectives
[5] antigout agents
[5] autonomic nervous system drugs
[6] neuromuscular blockers
[6] skeletal muscle relaxants
[5] blood drugs
[5] cardiovascular drugs
[6] antihypertensives
[5] central nervous system drugs
[6] analgesics and antipyretics
[6] antianxiety agents
[6] antidepressants
[6] antipsychotics
[6] narcotic and opioid analgesics
[6] nonsteroidal anti-inflammatory drugs
[6] sedative-hypnotics
[5] chemotherapeutics, antineoplastic agents
[5] dermatomucosal agents
[6] topical corticosteroids
[5] digestive system drugs
[6] antacids, adsorbents, and antiflatulents
[6] antidiarrheals
[6] antiemetics
[6] antiulcer agents
[6] digestants
[6] laxatives
[5] eye, ear, nose, and throat drugs
[6] nasal agents
[6] ophthalmics

```
                    [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