Contents

Send Us Your Comments ................................................................. xvii

Preface ............................................................................................... xix
  Audience ......................................................................................... xix
  Organization ................................................................................... xix
  Related Documentation ................................................................. xxii
  Conventions ................................................................................ xxiii
  Documentation Accessibility ....................................................... xxv

What's New in Oracle Text? ............................................................ xxvii
  Oracle Database 10g R1 New Features ........................................... xxvii
  Security Improvements ................................................................. xxvii
  Classification and Clustering ....................................................... xxvii
  Indexing ......................................................................................... xxviii
  Language Features ...................................................................... xxxi
  Querying ....................................................................................... xxxii
  Document Services ..................................................................... xxxiii

1 SQL Statements and Operators
  ALTER INDEX .................................................................................. 1-2
  ALTER TABLE: Supported Partitioning Statements ....................... 1-18
  CATSEARCH ................................................................................ 1-25
  CONTAINS .................................................................................. 1-34
  CREATE INDEX ........................................................................... 1-43
## 2 Oracle Text Indexing Elements

### Overview

- Creating Preferences .................................................. 2-2

### Datastore Types ......................................................... 2-3

- DIRECT_DATASTORE ................................................. 2-3
- MULTI_COLUMN_DATASTORE ................................... 2-4
- DETAIL_DATASTORE .................................................. 2-8
- FILE_DATASTORE ..................................................... 2-10
- URL_DATASTORE ...................................................... 2-12
- USER_DATASTORE .................................................... 2-15
- NESTED_DATASTORE ................................................. 2-18

### Filter Types ............................................................. 2-21

- CHARSET_FILTER .................................................. 2-21
- INSO_FILTER ......................................................... 2-23
- NULL_FILTER ........................................................ 2-27
- MAIL_FILTER ........................................................ 2-27
- USER_FILTER ........................................................ 2-30
- PROCEDURE_FILTER ............................................. 2-31

### Lexer Types ............................................................. 2-36

- BASIC_LEXER ....................................................... 2-36
- MULTI_LEXER ........................................................ 2-46
- CHINESE_VGRAM_LEXER ....................................... 2-48
- CHINESE_LEXER .................................................... 2-48
- JAPANESE_VGRAM_LEXER ....................................... 2-49
- JAPANESE_LEXER ................................................... 2-49
- KOREAN_LEXER ..................................................... 2-51
- KOREAN_MORPH_LEXER ......................................... 2-52
- USER_LEXER ........................................................ 2-56
- WORLD_LEXER ...................................................... 2-71

### Wordlist Type ............................................................ 2-72
BASIC_WORDLIST .................................................................................................................... 2-72
BASIC_WORDLIST Example ................................................................................................... 2-77
Storage Types ......................................................................................................................... 2-79
BASIC_STORAGE .................................................................................................................... 2-79
Section Group Types .......................................................................................................... 2-81
  Section Group Examples .................................................................................................. 2-82
Classifier Types .................................................................................................................. 2-84
  RULE_CLASSIFIER ......................................................................................................... 2-84
  SVM_CLASSIFIER ............................................................................................................. 2-85
Cluster Types ......................................................................................................................... 2-86
  KMEAN_CLUSTER ........................................................................................................... 2-87
  Hierarchical Clustering ................................................................................................. 2-88
Stoplists ................................................................................................................................. 2-89
  Multi-Language Stoplists ............................................................................................... 2-89
  Creating Stoplists .......................................................................................................... 2-89
  Modifying the Default Stoplist ...................................................................................... 2-90
System-Defined Preferences ........................................................................................... 2-90
  Data Storage .................................................................................................................. 2-91
  Filter ............................................................................................................................... 2-91
  Lexer ............................................................................................................................... 2-91
  Section Group ............................................................................................................... 2-93
  Stoplist ........................................................................................................................... 2-93
  Storage ............................................................................................................................ 2-94
  Wordlist .......................................................................................................................... 2-94
System Parameters .......................................................................................................... 2-94
  General System Parameters ......................................................................................... 2-94
  Default Index Parameters ............................................................................................ 2-95

3 CONTAINS Query Operators

Operator Precedence ........................................................................................................ 3-2
  Group 1 Operators ........................................................................................................ 3-2
  Group 2 Operators and Characters ............................................................................ 3-3
  Procedural Operators ................................................................................................ 3-3
  Precedence Examples .................................................................................................. 3-3
  Altering Precedence ..................................................................................................... 3-4
4 Special Characters in Queries

Grouping Characters .......................................................... 4-1
Escape Characters .......................................................... 4-2
  Querying Escape Characters .................................. 4-3
Reserved Words and Characters .................................. 4-3
5 CTX_ADM Package
RECOVER ......................................................................................................................... 5-2
SET_PARAMETER ........................................................................................................... 5-3

6 CTX_CLS Package
TRAIN ................................................................................................................................. 6-2
CLUSTERING .................................................................................................................. 6-6

7 CTX_DDL Package
ADD_ATTR_SECTION ....................................................................................................... 7-3
ADD_FIELD_SECTION .................................................................................................... 7-5
ADD_INDEX ...................................................................................................................... 7-9
ADD_MDATA .................................................................................................................... 7-11
ADD_MDATA_SECTION .................................................................................................... 7-14
ADD_SPECIAL_SECTION ................................................................................................. 7-16
ADD_STOPCLASS ........................................................................................................... 7-18
ADD_STOP_SECTION ...................................................................................................... 7-19
ADD_STOPHEME ............................................................................................................ 7-21
ADD_STOPWORD ........................................................................................................... 7-22
ADD_SUB_LEXER .......................................................................................................... 7-24
ADD_ZONE_SECTION ..................................................................................................... 7-27
CREATE_INDEX_SET ....................................................................................................... 7-31
CREATE_POLICY ............................................................................................................ 7-32
CREATE_PREFERENCE .................................................................................................. 7-35
CREATE_SECTION_GROUP ............................................................................................... 7-38
CREATE_STOPLIST ......................................................................................................... 7-41
DROP_INDEX_SET .......................................................................................................... 7-43
DROP_POLICY ............................................................................................................... 7-44
DROP_PREFERENCE ..................................................................................................... 7-45
DROP_SECTION_GROUP ................................................................................................. 7-46
DROP_STOPLIST ............................................................................................................. 7-47
OPTIMIZE_INDEX .......................................................................................................... 7-48
REMOVE_INDEX ............................................................................................................. 7-52
REMOVE_MDATA ........................................................................................................... 7-53
8 CTX_DOC Package
FILTER ................................................................................................................................. 8-3
GIST ..................................................................................................................................... 8-6
HIGHLIGHT ............................................................................................................... 8-11
IFILTER .................................................................................................................. 8-14
MARKUP .......................................................................................................................... 8-16
PKENCODE ....................................................................................................................... 8-22
POLICY_FILTER ..................................................................................................... 8-24
POLICY_GIST ....................................................................................................................... 8-25
POLICY_HIGHLIGHT .................................................................................................. 8-28
POLICY_MARKUP .................................................................................................... 8-30
POLICY_THEMES .............................................................................................................. 8-33
POLICY_TOKENS ............................................................................................................... 8-35
SET_KEY_TYPE ................................................................................................................ 8-37
THEMES .......................................................................................................................... 8-38
TOKENS ............................................................................................................................ 8-41

9 CTX_OUTPUT Package
ADD_EVENT ....................................................................................................................... 9-2
ADD_TRACE ...................................................................................................................... 9-3
END_LOG ......................................................................................................................... 9-5
END_QUERY_LOG ............................................................................................................ 9-6
GET_TRACE_VALUE ........................................................................................................ 9-7
LOG_TRACES .................................................................................................................. 9-8
LOGFILENAME ............................................................................................................... 9-9
10 **CTX_QUERY Package**

- BROWSE_WORDS .......................................................... 10-2
- COUNT_HITS ............................................................... 10-5
- EXPLAIN ................................................................. 10-6
- HFEEDBACK ............................................................... 10-9
- REMOVE_SQE ............................................................ 10-14
- STORE_SQE ............................................................... 10-15

11 **CTX_REPORT**

- Procedures in CTX_REPORT ........................................ 11-1
- Using the Function Versions ........................................ 11-2
- DESCRIBE_INDEX ...................................................... 11-3
- DESCRIBE_POLICY .................................................... 11-4
- CREATE_INDEX_SCRIPT .............................................. 11-5
- CREATE_POLICY_SCRIPT ............................................. 11-6
- INDEX_SIZE ............................................................ 11-7
- INDEX_STATS .......................................................... 11-9
- QUERY_LOG_SUMMARY .............................................. 11-14
- TOKEN_INFO ........................................................... 11-19
- TOKEN_TYPE ........................................................... 11-21

12 **CTX_THES Package**

- ALTER_PHRASE ......................................................... 12-3
- ALTER_THESAURUS .................................................... 12-5
- BT ............................................................................. 12-7
- BTG .......................................................................... 12-10
- BTI ........................................................................... 12-12
- BTP .......................................................................... 12-14
CREATE_PHRASE .......................................................................................................................... 12-16
CREATE_RELATION .......................................................................................................................... 12-18
CREATE_THESAURUS ......................................................................................................................... 12-20
CREATE_TRANSLATION ....................................................................................................................... 12-21
DROP_PHRASE .................................................................................................................................. 12-22
DROP_RELATION ................................................................................................................................. 12-24
DROP_THESAURUS ................................................................................................................................. 12-27
DROP_TRANSLATION ............................................................................................................................ 12-28
HAS_RELATION .................................................................................................................................. 12-29
NT ....................................................................................................................................................... 12-30
NTG ...................................................................................................................................................... 12-33
NTI ....................................................................................................................................................... 12-35
NTP ....................................................................................................................................................... 12-37
OUTPUT_STYLE ................................................................................................................................... 12-39
PT ....................................................................................................................................................... 12-40
RT ....................................................................................................................................................... 12-42
SN ....................................................................................................................................................... 12-44
SYN ..................................................................................................................................................... 12-45
THES_TT ............................................................................................................................................... 12-48
TR ....................................................................................................................................................... 12-49
TRSYN .................................................................................................................................................. 12-51
TT ....................................................................................................................................................... 12-53
UPDATE_TRANSLATION ....................................................................................................................... 12-55

13 CTX_ULEXER Package

WILDCARD_TAB ................................................................................................................................. 13-2

14 Executables

Thesaurus Loader (ctxload) .................................................................................................................. 14-1
  Text Loading ....................................................................................................................................... 14-1
txload Syntax ...................................................................................................................................... 14-2
txload Examples ................................................................................................................................... 14-4
Knowledge Base Extension Compiler (ctxkbtc) .................................................................................. 14-5
  Knowledge Base Character Set ......................................................................................................... 14-5
txkbtc Syntax...................................................................................................................................... 14-5
ctxkbtc Usage Notes ................................................................................................................... 14-6
ctxkbtc Limitations ...................................................................................................................... 14-7
ctxkbtc Constraints on Thesaurus Terms .................................................................................. 14-7
ctxkbtc Constraints on Thesaurus Relations ........................................................................... 14-7
Extending the Knowledge Base ................................................................................................. 14-8
Adding a Language-Specific Knowledge Base ......................................................................... 14-9
Order of Precedence for Multiple Thesauri .......................................................................... 14-10
Size Limits for Extended Knowledge Base ............................................................................. 14-10
Lexical Compiler (ctxlc) ........................................................................................................... 14-11
Syntax of ctxlc ........................................................................................................................... 14-11
Performance Considerations ................................................................................................... 14-12
ctxlc Usage Notes ...................................................................................................................... 14-12
Example ...................................................................................................................................... 14-12

15 Alternative Spelling

Overview of Alternative Spelling Features ........................................................................... 15-1
Alternate Spelling ..................................................................................................................... 15-2
Base-Letter Conversion ............................................................................................................. 15-2
New German Spelling ............................................................................................................... 15-3
Overriding Alternative Spelling Features ................................................................................ 15-4
Overriding Base-Letter Transformations with Alternate Spelling ........................................... 15-4
Alternative Spelling Conventions ........................................................................................... 15-4
German Alternate Spelling Conventions ................................................................................ 15-5
Danish Alternate Spelling Conventions .................................................................................. 15-5
Swedish Alternate Spelling Conventions ................................................................................ 15-5

A Result Tables

CTX_QUERY Result Tables ......................................................................................................... A-1
EXPLAIN Table ......................................................................................................................... A-1
HFEEDBACK Table ................................................................................................................... A-4
CTX_DOC Result Tables .......................................................................................................... A-7
Filter Table ............................................................................................................................... A-7
Gist Table ................................................................................................................................ A-7
Highlight Table ........................................................................................................................ A-8
Markup Table ............................................................................................................................ A-8
B  Supported Document Formats

About Document Filtering Technology ................................................................. B-1
  Latest Updates for Patch Releases ................................................................. B-2
  Supported Platforms ....................................................................................... B-2
  Environment Variables ................................................................................... B-3
  Requirements for UNIX Platforms ................................................................. B-3
Supported Document Formats ........................................................................... B-3
  Word Processing Formats - Generic Text ....................................................... B-3
  Word Processing Formats - DOS .................................................................. B-4
  Word Processing Formats - Windows ........................................................... B-5
  Word Processing Formats - Macintosh .......................................................... B-6
  Spreadsheet Formats ..................................................................................... B-6
  Database Formats ......................................................................................... B-7
  Display Formats ............................................................................................ B-8
  Presentation Formats ..................................................................................... B-8
  Graphic Formats ............................................................................................ B-9
  Other Document Formats ............................................................................... B-11
Restrictions on Format Support ....................................................................... B-11

C  Loading Examples

SQL INSERT Example ........................................................................................ C-1
SQL*Loader Example ....................................................................................... C-1
  Creating the Table ......................................................................................... C-2
  Issuing the SQL*Loader Command ............................................................... C-2
Structure of ctxload Thesaurus Import File ..................................................... C-4
  Alternate Hierarchy Structure ..................................................................... C-6
Usage Notes for Terms in Import Files .......................................................... C-7
Usage Notes for Relationships in Import Files .............................................. C-8
Examples of Import Files ............................................................................... C-9
D Multilingual Features

Introduction ........................................................................................................................................ D-1
Indexing ................................................................................................................................................ D-1
  Index Types ................................................................................................................................... D-1
  Lexer Types ................................................................................................................................... D-2
  Basic Lexer Features ..................................................................................................................... D-3
  Multi Lexer Features .................................................................................................................... D-5
  World Lexer Features ................................................................................................................... D-5
Querying ............................................................................................................................................ D-7
  ABOUT Operator .......................................................................................................................... D-8
  Fuzzy Operator ............................................................................................................................ D-8
  Stem Operator ............................................................................................................................... D-8
Supplied Stop Lists .......................................................................................................................... D-8
Knowledge Base .............................................................................................................................. D-8
  Knowledge Base Extension ........................................................................................................... D-8
Multi-Lingual Features Matrix ........................................................................................................ D-9

E Supplied Stoplists

English Default Stoplist .................................................................................................................. E-1
Chinese Stoplist (Traditional) ........................................................................................................ E-2
Chinese Stoplist (Simplified) ......................................................................................................... E-2
Danish (dk) Default Stoplist ......................................................................................................... E-3
Dutch (nl) Default Stoplist ........................................................................................................... E-3
Finnish (sf) Default Stoplist .......................................................................................................... E-4
French (f) Default Stoplist ............................................................................................................ E-5
German (d) Default Stoplist .......................................................................................................... E-6
Italian (i) Default Stoplist ............................................................................................................ E-7
Portuguese (pt) Default Stoplist .................................................................................................. E-8
Spanish (e) Default Stoplist .......................................................................................................... E-8
Swedish (s) Default Stoplist .......................................................................................................... E-9

F Scoring Algorithm

Scoring Algorithm for Word Queries .......................................................................................... F-1
  Example.......................................................................................................................................... F-2
DML and Scoring

G  Views

CTX_CLASSES.......................................................................................................................... G-3
CTX_INDEXES.......................................................................................................................... G-3
CTX_INDEX_ERRORS................................................................................................................ G-4
CTX_INDEX_OBJECTS................................................................................................................ G-4
CTX_INDEX_PARTITIONSW ........................................................................................................ G-4
CTX_INDEX_SETS........................................................................................................................ G-4
CTX_INDEX_SET_INDEXES......................................................................................................... G-5
CTX_INDEX_SUB_LexERS .......................................................................................................... G-5
CTX_INDEX_SUB_LEXER_VALUES ............................................................................................. G-6
CTX_INDEX_VALUES................................................................................................................ G-6
CTX_OBJECTS........................................................................................................................... G-7
CTX_OBJECT_ATTRIBUTE_ATTRIBUTES..................................................................................... G-7
CTX_OBJECT_ATTRIBUTE_LOV ................................................................................................... G-8
CTX_PARAMETERS..................................................................................................................... G-8
CTX_PENDING ......................................................................................................................... G-10
CTX_PREFERENCES................................................................................................................... G-10
CTX_PREFERENCE_VALUES ..................................................................................................... G-10
CTX_SECTIONS........................................................................................................................ G-11
CTX_SECTION_GROUPS.............................................................................................................. G-11
CTX_SQES.................................................................................................................................. G-11
CTX_STOPLISTS......................................................................................................................... G-12
CTX_STOPWORDS...................................................................................................................... G-12
CTX_SUB_LexERS ..................................................................................................................... G-12
CTX_THESAURI .......................................................................................................................... G-13
CTX_THES_PHRASES ................................................................................................................ G-13
CTX_TRACE_VALUES ................................................................................................................ G-13
CTX_USER_INDEXES .................................................................................................................. G-14
CTX_USER_INDEX_ERRORS ....................................................................................................... G-15
CTX_USER_INDEX_OBJECTS ....................................................................................................... G-15
CTX_USER_INDEX_PARTITIONSW ............................................................................................... G-16
CTX_USER_INDEX_SETS.............................................................................................................. G-17
CTX_USER_INDEX_SET_INDEXES ............................................................................................... G-17
H  Stopword Transformations

Understanding Stopword Transformations .................................................. H-1
    Word Transformations........................................................................ H-2
    AND Transformations .................................................................... H-2
    OR Transformations ...................................................................... H-3
    ACCUMulate Transformations ....................................................... H-3
    MINUS Transformations ................................................................ H-4
    NOT Transformations ..................................................................... H-4
    EQUIValence Transformations ....................................................... H-4
    NEAR Transformations .................................................................. H-5
    Weight Transformations .................................................................. H-5
    Threshold Transformations ............................................................. H-6
    WITHIN Transformations ............................................................... H-6

Index
Send Us Your Comments

Oracle Text Reference, 10g Release 1 (10.1)
Part No. B10730-01

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 title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

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

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

If you have problems with the software, please contact your local Oracle Support Services.
This manual provides reference information for Oracle Text. Use it as a reference for creating Oracle Text indexes, for issuing Oracle Text queries, for presenting documents, and for using the Oracle Text PL/SQL packages.

This preface contains these topics:

- **Audience**
- **Organization**
- **Related Documentation**
- **Conventions**
- **Documentation Accessibility**

### Audience

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

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

If you are unfamiliar with the Oracle RDBMS and related tools, see the *Oracle Database Concepts*, which is a comprehensive introduction to the concepts and terminology used throughout Oracle documentation.

### Organization

This document contains:
Chapter 1, "SQL Statements and Operators"
This chapter describes the SQL statements and operators you can use with Oracle Text.

Chapter 2, "Oracle Text Indexing Elements"
This chapter describes the indexing types you can use to create an Oracle Text index.

Chapter 3, "CONTAINS Query Operators"
This chapter describes the operators you can use in CONTAINS queries.

Chapter 4, "Special Characters in Queries"
This chapter describes the special characters you can use in CONTAINS queries.

Chapter 5, "CTX_ADM Package"
This chapter describes the procedures in the CTX_ADM PL/SQL package.

Chapter 6, "CTX_CLS Package"
This chapter describes the procedures in the CTX_CLS PL/SQL package.

Chapter 7, "CTX_DDL Package"
This chapter describes the procedures in the CTX_DDL PL/SQL package. Use this package for maintaining your index.

Chapter 8, "CTX_DOC Package"
This chapter describes the procedures in the CTX_DOC PL/SQL package. Use this package for document services such as document presentation.

Chapter 9, "CTX_OUTPUT Package"
This chapter describes the procedures in the CTX_OUTPUT PL/SQL package. Use this package to manage your index error log files.

Chapter 10, "CTX_QUERY Package"
This chapter describes the procedures in the CTX_QUERY PL/SQL package. Use this package to manage queries such as to count hits and to generate query explain plan information.
Chapter 11, "CTX_REPORT"
This chapter describes the procedures in the CTX_REPORT PL/SQL package. Use this package to create various index reports.

Chapter 12, "CTX_THES Package"
This chapter describes the procedures in the CTX_THES PL/SQL package. Use this package to manage your thesaurus.

Chapter 13, "CTX_ULEXER Package"
This chapter describes the data types in the CTX_ULEXER PL/SQL package. Use this package with the user defined lexer.

Chapter 14, "Executables"
This chapter describes the supplied executables for Oracle Text including ctxload, the thesaurus loading program, and ctxkbtc, the knowledge base compiler.

Chapter 15, "Alternative Spelling"
This chapter describes how to handle terms that have multiple spellings, and it lists the alternate spelling conventions used for German, Danish, and Swedish.

Appendix A, "Result Tables"
This appendix describes the result tables for some of the procedures in CTX_DOC, CTX_QUERY, and CTX_THES packages.

Appendix B, "Supported Document Formats"
This appendix describes the supported document formats that can be filtered with the Inso filter for indexing.

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

Chapter D, "Multilingual Features"
This appendix describes the multilingual features of Oracle Text.

Appendix E, "Supplied Stoplists"
This appendix describes the supplied stoplist for each supported language.
Appendix F, "Scoring Algorithm"
This appendix describes the scoring algorithm used for word queries.

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

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

Related Documentation
For more information, see these Oracle resources:

For more information about Oracle Text, see:

- Oracle Text Application Developer’s Guide

For more information about Oracle Database, see:

- Oracle Database Concepts
- Oracle Database Administrator’s Guide
- Oracle Database Utilities
- Oracle Database Performance Tuning Guide
- Oracle Database SQL Reference
- Oracle Database Reference
- Oracle Database Application Developer’s Guide - Fundamentals

For more information about PL/SQL, see:

- PL/SQL User’s Guide and Reference

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

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

Many books in the documentation set use the sample schemas of the seed database, which is installed by default when you install Oracle Database. Refer to Oracle Database Sample Schemas for information on how these schemas were created and how you can use them yourself.
Printed documentation is available for sale in the Oracle Store at

http://oraclestore.oracle.com/

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

http://otn.oracle.com/membership/

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

http://otn.oracle.com/documentation/

Conventions

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

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

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

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.</td>
<td>The C datatypes such as <strong>ub4</strong>, <strong>sword</strong>, or <strong>OCINumber</strong> are valid. When you specify this clause, you create an index-organized table.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Italic typeface indicates query terms, book titles, emphasis, syntax clauses, or placeholders.</td>
<td>The following query searches for <em>oracle</em>. <em>Oracle Database Concepts</em> You can specify the <em>parallel_clause</em>. Run <code>old_release.SQL</code> where <em>old_release</em> refers to the release you installed prior to upgrading.</td>
</tr>
</tbody>
</table>
### Conventions in Code Examples

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

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

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

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Uppercase monospace typeface indicates elements supplied by the system.</td>
<td>You can specify this clause only for a NUMBER column.</td>
</tr>
<tr>
<td>monospace</td>
<td></td>
<td>You can back up the database using the BACKUP command.</td>
</tr>
<tr>
<td>(fixed-width</td>
<td></td>
<td>Query the TABLE_NAME column in the USER_TABLES data dictionary view.</td>
</tr>
<tr>
<td>font)</td>
<td></td>
<td>Specify the ROLLBACK_SEGMENTS parameter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use the DBMS_STATS.GENERATE_STATS procedure.</td>
</tr>
<tr>
<td>lowercase</td>
<td>Lowercase monospace typeface indicates executables and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, user names and roles, program units, and parameter values.</td>
<td>Enter sqlplus to open SQL*Plus.</td>
</tr>
<tr>
<td>monospace</td>
<td></td>
<td>The department_id, department_name, and location_id columns are in the hr.departments table.</td>
</tr>
<tr>
<td>(fixed-width</td>
<td></td>
<td>Set the QUERY_REWRITE_ENABLED initialization parameter to true.</td>
</tr>
<tr>
<td>font)</td>
<td></td>
<td>Connect as oe user.</td>
</tr>
</tbody>
</table>

### Conventions

- **Brackets** enclose one or more optional items. Do not enter the brackets.

  - `DECIMAL (digits [ , precision ])

- **Braces** enclose two or more items, one of which is required. Do not enter the braces.

  - `{ENABLE | DISABLE}

- **Vertical bar** represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.

  - `{ENABLE | DISABLE}
  
  - `COMPRESS | NOCOMPRESS`
Documentation Accessibility

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

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Horizontal ellipsis points indicate either:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- That we have omitted parts of the code that are not directly related to the example</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- That you can repeat a portion of the code</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.</td>
<td></td>
</tr>
<tr>
<td>Other notation</td>
<td>You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as it is shown.</td>
<td>acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;</td>
</tr>
<tr>
<td>Italics</td>
<td>Italicized text indicates variables for which you must supply particular values.</td>
<td>CONNECT SYSTEM/system_password</td>
</tr>
<tr>
<td>UPPERCASE</td>
<td>Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.</td>
<td>SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;</td>
</tr>
<tr>
<td>lowercase</td>
<td>Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.</td>
<td>SELECT last_name, employee_id FROM employees; sqlplus hr/hr</td>
</tr>
</tbody>
</table>

**Documentation Accessibility**

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

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

This chapter describes new features of Oracle Text and provides pointers to additional information.

Oracle Database 10g R1 New Features

The following features are new for this release:

Security Improvements

In previous versions of Oracle Text, CTXSYS had DBA privileges. To tighten security and protect the database in the case of unauthorized access, CTXSYS now has only CONNECT and RESOURCE roles, and only limited, necessary direct grants on some system views and packages. Some applications using Oracle Text may therefore require minor changes in order to work properly with this security change.

See Also: The Migration chapter in the Oracle Text Application Developer’s Guide

Classification and Clustering

The following features are new for classification and clustering:

- Supervised Training and Document Classification

  TheCTX_CLS.TRAIN procedure has been enhanced to support an additional classifier type called Support Vector Machine method for the supervised training of documents. The SVM method of training can produce better rules for classification than the query-based method.
Document Clustering

The new CTX_CLS.CLUSTERING procedure enables you to generate document clusters. A cluster is a group of documents similar to each other in content.

See Also: CLUSTERING in Chapter 6, "CTX_CLS Package" and the Oracle Text Application Developer’s Guide

Indexing

The following features are new for indexing.

- Automatic and ON COMMIT Synchronization for CONTEXT index
  You can set the CONTEXT index to synchronize automatically either at intervals you specify or at commit time.

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

- Transactional CONTEXT Indexes
  The new TRANSACTIONAL parameter to CREATE INDEX and ALTER INDEX enables changes to a base table to be immediately queryable.

  See Also: TRANSACTIONAL in SQL Statements and Operators

- Automatic Multi-Language Indexing
  The new WORLD_LEXER lexer type includes automatic language detection in documents, enabling you to index multilingual documents without having to include a language column in a base table.

  See Also: WORLD_LEXER in Chapter 2, "Oracle Text Indexing Elements"

- Mail Filtering
  Oracle Text can filter and index RFC-822 email messages. To do so, you use the new MAIL_FILTER filter preference.
Fast Filtering of Binary Documents

New attributes for the **INSO_FILTER** and **MAIL_FILTER** filter preferences offer the option of significantly improving performance when filtering binary documents. This fast filtering preserves only a limited amount of document formatting.

**See Also:**  **INSO_FILTER** and **MAIL_FILTER** in Chapter 2, "Oracle Text Indexing Elements"

Support for creating local partitioned **CONTEXT** indexes in parallel

You can now create local partitioned **CONTEXT** indexes in parallel with **CREATE INDEX**.

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

**MDATA** section for adding metadata to documents

You can now add an **MDATA** section to a section group. **MDATA** sections define metadata that enables you to perform mixed **CONTAINS** queries faster.

**See Also:**  **ADD_MDATA** and **ADD_MDATA_SECTION** in Chapter 7, "CTX_DDL Package"; **MDATA** in Chapter 3, "CONTAINS Query Operators"; the section searching chapter in the Oracle Text Application Developer’s Guide

**ALTER TABLE** enhanced support for partitioned tables

**ALTER TABLE** supports the **UPDATE GLOBAL INDEXES** clause for partitioned tables.

**See Also:**  **ALTER TABLE: Supported Partitioning Statements** in Chapter 1, "SQL Statements and Operators"
### Binary Filtering for MULTI_COLUMN_DATASTORE

The `MULTI_COLUMN_DATASTORE` now enables you to filter binary columns into text for concatenation with other columns during indexing. This datastore has also been enhanced to switch its XML-like auto-tagging on and off.

**See Also:** `MULTI_COLUMN_DATASTORE` in Chapter 2, "Oracle Text Indexing Elements"

### New XML Output Option for Index Reports

Several procedures and functions in the `CTX_REPORT` package now include a `report_format` parameter that enables you to obtain index report output either as plain text or XML.

**See Also:** Chapter 11, "CTX_REPORT"

### Replacing Index Metadata

You can replace index metadata (preference attributes) without having to rebuild the index. You do this using the new `METADATA` keyword with `ALTER INDEX`.

**See Also:** `ALTER INDEX REBUILD` Syntax in Chapter 1, "SQL Statements and Operators"

### New Columns for Oracle Text Views

Three Oracle Text views, `CTX_OBJECT_ATTRIBUTES`, `CTX_INDEX_PARTITIONS`, and `CTX_USER_INDEX_PARTITIONS`, have new columns.

**See Also:** Appendix G, "Views"

### New Options for Index Optimization

`CTX_DDL.OPTIMIZE_INDEX` has two new optlevels. `TOKEN_TYPE` optimizes on demand all tokens in the index matching the input token type. This is intended to help users keep critical field sections or `MDATA` sections optimal. `REBUILD` enables `CTX_DDL.OPTIMIZE_INDEX` to rebuild an index entirely.

**See Also:** `OPTIMIZE_INDEX` in Chapter 7, "CTX_DDL Package"
Log tokens During Index Optimization

The `CTX_OUTPUT.EVENT_OPT_PRINT_TOKEN` event, which prints each token as it is being optimized, can be used with `CTX_OUTPUT.ADD_EVENT`.

**See Also:**  `ADD_EVENT` in Chapter 9, "CTX_OUTPUT Package"

Tracing

Oracle Text includes a tracing facility that enables you to identify bottlenecks in indexing and querying.

**See Also:**  `ADD_TRACE` in Chapter 9, "CTX_OUTPUT Package"  
and the Oracle Text Application Developer’s Guide

New German Spelling

Oracle Text now can index German words under both traditional and reformed spelling.

**See Also:**  New German Spelling in Chapter 15, "Alternative Spelling"

Language Features

The following are new language features:

- Japanese Language Enhancements
  Oracle Text supports stem queries in Japanese with the stem `$` operator.
  **See Also:**  `BASIC_WORDLIST` in Chapter 2, "Oracle Text Indexing Elements"  
  stem (`$`) operator in Chapter 3, "CONTAINS Query Operators"

- Customization of Japanese and Chinese Lexicons
  A new command, `ctxlc`, enables you to either modify the existing system Japanese and Chinese dictionaries (lexicons) or create new dictionaries from the merging of the system dictionaries with user-provided word lists.  `ctxlc` also outputs the contents of dictionaries as word files.
  **See Also:**  Lexical Compiler (`ctxlc`) in Chapter 14, "Executables"
New character sets for the Chinese VGRAM lexer

The Chinese VGRAM lexer now supports the AL32UTF8 and ZHS32GB18030 character sets.

See Also: CHINESE_VGRAM_LEXER in Chapter 2, “Oracle Text Indexing Elements”

Querying

Query Template Enhancements

Query templating has been enhanced to provide the following features:

- progressive relaxation of queries, which enables you to progressively execute less restrictive versions of a single query
- query rewriting, which enables you to programatically rewrite any single query into different versions to increase recall
- query language specification
- alternative scoring algorithms

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

The Querying chapter in the Oracle Text Application Developer’s Guide

Query Log Analysis

Oracle Text now offers the capability to create a log of queries and to issue reports on its contents, indicating, for example, the most or least frequent successful queries.

See Also:

QUERY_LOG_SUMMARY in Chapter 11, "CTX_REPORT"

START_QUERY_LOG and END_QUERY_LOG in Chapter 9, "CTX_OUTPUT Package"

XML DB Enhancements

Oracle Text has the following XML DB enhancements:
Better performance of existsNode() / CTXXPATH queries, with new support for attribute existence searching, and positional predicates.

Support for positional predicate testing with INPATH and HASPATH operators

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

*Oracle XML DB Developer’s Guide*

Overriding of Base-letter Transformations

A new BASIC_LEXER attribute, OVERRIDE_BASE_LETTER, prevents unexpected results when base-letter transformations are combined with alternate spelling.

**See Also:** Overview of Alternative Spelling Features in Chapter 15, "Alternative Spelling"

Document Services

**Highlighting with INPATH and HASPATH**

Oracle Text supports highlighting with INPATH and HASPATH operators.

**See Also:** Chapter 8, "CTX_DOC Package"

**CTX_DOC Enhancements for Policy-Based Document Services**

With the new CTX_DOC.POLICY_* procedures, you can perform document highlighting and filtering without requiring a table or a context index.

**See Also:** Chapter 8, "CTX_DOC Package"
This chapter describes the SQL statements and Oracle Text operators you use for creating and managing Text indexes and performing Text queries.

The following statements are described in this chapter:

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

---

**Purpose**

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

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

**CONTEXT and CTXRULE Indextypes**

You can use `ALTER INDEX` to perform the following task on `CONTEXT` and `CTXRULE` indextypes:

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

**See Also:** `ALTER INDEX REBUILD Syntax` to learn more about performing these tasks.

---

**Note:** This section describes the `ALTER INDEX` statement as it pertains to managing a Text domain index.

For a complete description of the `ALTER INDEX` statement, see *Oracle Database SQL Reference*.
ALTER INDEX RENAME Syntax

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

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

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

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

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

---

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

- **part_name**
  Specify the name of the index partition to rename.

- **new_part_name**
  Specify the new name for partition.

ALTER INDEX REBUILD Syntax

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

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

- **PARTITION partname**
  Rebuilds the index partition `partname`. Only one index partition can be built at a time.

  When you rebuild a partition you can specify only SYNC, OPTIMIZE FULL/FAST, RESUME, or REPLACE in `paramstring`. These operations work only on the `partname`
you specify. You cannot specify RESUME when you rebuild partitions or a partitioned index.

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

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

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

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

ONLINE enables you to continue to perform updates, inserts, and deletes on a base table; it does not enable you to query the base table.

You cannot use PARALLEL with ONLINE. ONLINE is only supported for CONTEXT indexes.

---

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

---

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

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

You cannot use PARALLEL with ONLINE.
ALTER INDEX

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

The syntax for paramstring is as follows:

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

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

    | RESUME [memory memsize]
    | OPTIMIZE [token index_token | fast | full [maxtime (time | unlimited)]
    | SYNC [memory memsize]
    | ADD STOPWORD word [language language]
    | ADD ZONE SECTION section_name tag tag
    | ADD FIELD SECTION section_name tag tag ((VISIBLE | INVISIBLE))
    | ADD ATTR SECTION section_name tag tag@attr
    | ADD STOP SECTION tag'

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

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

If you are rebuilding a partitioned index with REPLACE, you can only specify STORAGE and MEMORY.
**See Also:** Chapter 2, "Oracle Text Indexing Elements" for more information about creating and setting preferences, including information about system-defined preferences.

**REPLACE METADATA** preference new_preference

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

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

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

`ALTER INDEX REBUILD PARAMETER ('REPLACE METADATA')` does not work for a local partitioned index at the index (global) level; you cannot, for example, use this syntax to change a global preference, such as filter or lexer type, without rebuilding the index. Use `CTX_DDL.REPLACE_INDEX_METADATA` instead.

**When is the METADATA keyword ok to use?**

This command is intended only for when the change in index metadata would not lead to an inconsistent index, which can lead to incorrect query results.

For example, you can use this command in the following instances:

- to go from a single-language lexer to a multi-lexer in anticipation of multi-lingual data. For an example, see "Replacing Index Metadata: Changing Single-lexer to Multi-lexer" on page 1-15.
- to change the WILDCARD_MAXTERMS setting in BASIC_WORDLIST.
- to change the type of SYNC, which can be automatic, manual, or on-commit.

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

**Caution:** The REPLACE METADATA command can result in inconsistent index data, which can lead to incorrect query results. As such, Oracle does not recommend using this command, unless you carefully consider the effect it will have on the consistency of your index data and subsequent queries.
There can be many instances when changing metadata can result in inconsistent index data. For example, Oracle does not advise you to use the METADATA keyword after doing the following:

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

In these unsafe cases, Oracle recommends rebuilding the index.

**[METADATA] REPLACE SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)**

Specify SYNC for automatic synchronization of the CONTEXT index when there is DML to the base table. You can specify one of the following SYNC methods:

<table>
<thead>
<tr>
<th>SYNC type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MANUAL</td>
<td>No automatic synchronization. This is the default. You must manually synchronize the index with CTX_DDL.SYNC_INDEX. Use MANUAL to disable ON COMMIT and EVERY synchronization.</td>
</tr>
<tr>
<td>EVERY interval-string</td>
<td>Automatically synchronize the index at a regular interval specified by the value of interval-string. interval-string takes the same syntax as that for scheduler jobs. Automatic synchronization using EVERY requires that the index creator have CREATE JOB privileges. Make sure that interval-string is set to a long enough period that any previous sync jobs will have completed; otherwise, the sync job may hang. interval-string must be enclosed in double quotes.</td>
</tr>
</tbody>
</table>

See Enabling Automatic Index Synchronization on page 1-54 for an example of automatic sync syntax.
Each partition of a locally partitioned index can have its own type of sync (ON COMMIT, EVERY, or MANUAL). The type of sync specified in master parameter strings applies to all index partitions unless a partition specifies its own type.

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

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

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

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

[METADATA] TRANSACTIONAL | NONTRANSACTIONAL

This parameter enables you to turn the TRANSACTIONAL property on or off. For more on TRANSACTIONAL, see "TRANSACTIONAL" on page 1-52 in this book.

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

To turn on TRANSACTIONAL index property:

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

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

To turn off TRANSACTIONAL index property:

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

or

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

RESUME [MEMORY memsize]

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

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

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

Note: This ALTER INDEX operation will not be supported in future releases.

To optimize your index, use CTX_DDL.OPTIMIZE_INDEX.

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

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

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

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

Note: Optimizing in full mode runs even when there are no deleted document rows. This is useful when you need to optimize time-limited batches with the maxtime parameter.
You use the `maxtime` parameter to specify in minutes the time Oracle Text is to spend on the optimization operation. Oracle Text starts the optimization where it left off and optimizes until complete or until the time limit has been reached, whichever comes first. Specifying a time limit is useful for automating index optimization, where you set Oracle Text to optimize the index for a specified time on a regular basis.

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

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

---

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

---

**SYNC [MEMORY `memsize`]**

---

**Note:** This `ALTER INDEX` operation will not be supported in future releases.

To synchronize your index, use `CTX_DDL.SYNC_INDEX`.

---

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

---

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

---

**Memory Considerations**

The memory parameter `memsize` specifies the amount of memory Oracle Text uses for the `ALTER INDEX` operation before flushing the index to disk. Specifying a large amount of memory improves indexing performance because there is less I/O and improves query performance and maintenance because there is less fragmentation.
Specifying smaller amounts of memory increases disk I/O and index fragmentation, but might be useful if you want to track indexing progress or when run-time memory is scarce.

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

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

When your stoplist is a multi-language stoplist, you must specify language.

The index is *not* rebuilt by this statement.

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

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

The index is *not* rebuilt by this statement.

---

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

---

**See Also:** "Add Section Constraints” on page 1-13

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

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

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

The added section section_name applies only to documents indexed after this operation. For the change to affect previously indexed documents, you must explicitly re-index the documents that contain the tag.
The index is *not* rebuilt by this statement.

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

**See Also:** "Add Section Constraints" 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 statement.

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

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

**ADD STOP SECTION** `tag`

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

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

The text within a stop section is always searchable.

The number of stop sections you can add is unlimited.

The index is *not* rebuilt by this statement.
ALTER INDEX

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

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

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

ALTER INDEX Examples

Resuming Failed Index
The following statement resumes the indexing operation on newsindex with 2 megabytes of memory:

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

Rebuilding an Index
The following statement rebuilds the index, replacing the stoplist preference with new_stop.

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

Rebuilding a Partitioned Index
The following example creates a partitioned text table, populates it, and creates a partitioned index. It then adds a new partition to the table and then rebuilds the index with ALTER INDEX:
PROMPT create partitioned table and populate it

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

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

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

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

The following statement rebuilds the index in the newly populated partition. In
general, the index partition name for a newly added partition is the same as the
table partition name, unless it is already been used. In this case, Oracle Text
generates a new name.
alter index part_idx rebuild partition p_tab4;

The following statement queries the table for the two hits in the newly added
partition:
select * from part_tab where contains(b,'passiflora') >0;

The following statement queries the newly added partition directly:
select * from part_tab partition (p_tab4) where contains(b,'passiflora') >0;
Replacing Index Metadata: Changing Single-lexer to Multi-lexer

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

*REM create a simple table, which stores only english (American) text*

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

*REM we’ll create a simple lexer to lex this english text*

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

*REM create a text index on the simple table*

```sql
create index simple_idx on simple(text)
indextype is ctxsys.context parameters ('lexer us_lexer');
```

*REM we can query easily*

```sql
select * from simple where contains(text, 'fox')>0;
```

*REM now suppose we want to start accepting spanish documents.*

*REM first we have to extend the table with a language column*

```sql
alter table simple add (lang varchar2(10) default 'us');
```

*REM now let's create a spanish lexer,*

```sql
begin
    ctx_ddl.create_preference('e_lexer','basic_lexer');
    ctx_ddl.set_attribute('e_lexer','base_letter','yes');
end;
/
```

*REM Then we create a multi-lexer incorporating our english and spanish lexers.*

*REM Note that the DEFAULT lexer is the exact same lexer that we have already*  
*REM indexed all the documents with.*

```sql
begin
    ctx_ddl.create_preference('m_lexer','multi_lexer');
    ctx_ddl.add_sub_lexer('m_lexer','default','us_lexer');
    ctx_ddl.add_sub_lexer('m_lexer','spanish','e_lexer');
end;
/
```

*REM now let's replace our metadata*
alter index simple_idx rebuild
parameters ('replace metadata language column lang lexer m_lexer');

REM we're ready for some spanish data. Note that we could have inserted
REM this BEFORE the alter index, as long as we didn't SYNC.
insert into simple values ('el zorro marrón rápido', 'e');
commit;
exec ctx_ddl.sync_index('simple_idx');
REM now we can query the spanish data with base lettering:
select * from simple where contains(text, 'rapido')>0;

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

Synchronizing the Index
Synchronizing the index with ALTER INDEX will not be supported in future
releases. To synchronize your index, use CTX_DDL.SYNC_INDEX.

Adding a Zone Section
To add to the index the zone section author identified by the tag <author>, issue
the following statement:

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

Adding a Stop Section
To add a stop section identified by tag <fluff> to the index that uses the AUTO_SECTION_GROUP, issue the following statement:

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

Adding an Attribute Section
Assume that the following text appears in an XML document:

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

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

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

Related Topics

CTX_DDL.SYNC_INDEX in Chapter 7, "CTX_DDL Package"
CTX_DDL.OPTIMIZE_INDEX in Chapter 7, "CTX_DDL Package"
CREATE INDEX
Purpose

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

Modify Partition Syntax

**Unusable Local Indexes**

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

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

If the index partition is not marked unusable, the rebuild command returns without actually rebuilding the local index partition.

**Rebuild Unusable Local Indexes**

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

Rebuilds the index partition corresponding to the specified table partition that has an UNUSABLE status.
ALTER TABLE [schema.]table ADD PARTITION [partition]
VALUES LESS THAN (value_list) [partition_description]

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

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

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

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

Merge Partition Syntax

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

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

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

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

**Split Partition Syntax**

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

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

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

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

**Exchange Partition Syntax**

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

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

If EXCLUDING INDEXES is specified, all the context indexes corresponding to the partition and all the indexes on the exchanged table are marked as UNUSABLE. To rebuild the new index partition this case, you can issue ALTER TABLE MODIFY PARTITION.
If **INCLUDING INDEXES** is specified, then for every local domain index on the partitioned table, there must be a non-partitioned domain index on the non-partitioned table. The local index partitions are exchanged with the corresponding regular indexes.

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

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

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

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

**Restrictions**
Both indexes must be equivalent. They must use the same objects, same settings for each object. Note: we only check that they are using the same object. But they should use the same exact everything.

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

If either index or index partition does not meet all these restrictions an error is raised and both the index and index partition will be **INVALID**. The user needs to manually rebuild both index and index partition using **ALTER INDEX REBUILD**.

**Truncate Partition Syntax**

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

Removes all rows from a partition in a table. Corresponding CONTEXT index partitions are also removed.
For a global index, if you perform the truncate operation without an `UPDATE GLOBAL INDEXES` clause, the resulting index (if not NULL) will be invalid and must be rebuilt. If you specify the `UPDATE GLOBAL INDEXES` clause after the operation, the index will be valid.

**ALTER TABLE Examples**

**Global Index on Partitioned Table Examples**

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

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

insert into tdrexglb_part values (1,'row1');
insert into tdrexglb_part values (8,'row2');
insert into tdrexglb_part values (11,'row11');
insert into tdrexglb_part values (18,'row18');
insert into tdrexglb_part values (21,'row21');
insert into tdrexglb_part values (28,'row28');
commit;
create index tdrexglb_parti on tdrexglb_part(b) indextype is ctxsys.context;

create table tdrexglb(a int, b varchar2(40));
insert into tdrexglb values(20,'newrow20');
commit;

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

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

PROMPT before sync
select * from tdrexglb_part where contains(b,'row11') >0;
select * from tdrexglb_part where contains(b,'row18') >0;
```
exec ctx_dml.sync_index('tdrexglb_parti')

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

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

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

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

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

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

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

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

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

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

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

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

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

update global indexes;
CATSEARCH

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

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

About Performance

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

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

Limitation

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

Syntax

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

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

text_query
Specify one of the following to define your search in column.

- CATSEARCH query operations
Query Template Specification (for using CONTEXT grammar)

CATSEARCH query operations

The CATSEARCH operator supports only the following query operations:

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

These operators have the following syntax:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntax</th>
<th>Description of Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical AND</td>
<td>a b c</td>
<td>Returns rows that contain a, b and c.</td>
</tr>
<tr>
<td>Logical OR</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>Logical NOT</td>
<td>a - b</td>
<td>Returns rows that contain a and not b.</td>
</tr>
<tr>
<td>hyphen with no space</td>
<td>a-b</td>
<td>Hyphen treated as a regular character. For example, if the hyphen is defined as skipjoin, words such as web-site are treated as the single query term website. Likewise, if the hyphen is defined as a printjoin, words such as web-site are treated as web-site in the CTXCAT query language.</td>
</tr>
<tr>
<td>&quot; &quot;</td>
<td>&quot;a b c&quot;</td>
<td>Returns rows that contain the phrase &quot;a b c&quot;. For example, entering &quot;Sony CD Player&quot; means return all rows that contain this sequence of words.</td>
</tr>
<tr>
<td>()</td>
<td>(A B)</td>
<td>C</td>
</tr>
</tbody>
</table>
The following limitations apply to these operators:

- The left-hand side (the column name) must be a column named in at least one of the indexes of the index set.
- The left-hand side must be a plain column name. Functions and expressions are not allowed.
- The right-hand side must be composed of literal values. Functions, expressions, other columns, and subselects are not allowed.
- Multiple criteria can be combined with \texttt{AND}. \texttt{OR} is not supported.

For example, these expressions are supported:

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

And these expression are not supported:

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

**Query Template Specification**

You specify a marked-up string that specifies a query template. You can specify one of the following templates:

- query rewrite, used to expand a query string into different versions
progressive relaxation, used to progressively issue less restrictive versions of a query to increase recall

- alternate grammar, used to specify CONTAINS operators (See CONTEXT Query Grammar Examples)

- alternate language, used to specify alternate query language

- alternate scoring, used to specify alternate scoring algorithms

**See Also:** text_query parameter description for CONTAINS for more information about the syntax for these query templates.

**structured_query**

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

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

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

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

**Examples**

1. Create the Table

The following statement creates the table to be indexed.
CREATE TABLE auction (category_id number primary key, title varchar2(20),
bid_close date);

The following table inserts the values into the table:

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

1. Create the CTXCAT Index

The following statements create the CTXCAT index:

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

1. Query the Table

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

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

<table>
<thead>
<tr>
<th>CATEGORY_ID</th>
<th>TITLE</th>
<th>BID_CLOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Canon digital camera</td>
<td>26-FEB-00</td>
</tr>
<tr>
<td>7</td>
<td>Nikon digital camera</td>
<td>22-FEB-00</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>CATEGORY_ID</th>
<th>TITLE</th>
<th>BID_CLOSE</th>
</tr>
</thead>
</table>

SQL Statements and Operators  1-29
The following query finds all rows with the terms Sony and CD and Player:

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

<table>
<thead>
<tr>
<th>CATEGORY_ID</th>
<th>TITLE</th>
<th>BID_CLOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Sony CD Player</td>
<td>25-FEB-00</td>
</tr>
<tr>
<td>2</td>
<td>Sony CD Player</td>
<td>24-FEB-00</td>
</tr>
<tr>
<td>1</td>
<td>Sony CD Player</td>
<td>20-FEB-00</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>CATEGORY_ID</th>
<th>TITLE</th>
<th>BID_CLOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Tascam CD Burner</td>
<td>25-FEB-00</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>CATEGORY_ID</th>
<th>TITLE</th>
<th>BID_CLOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Pioneer DVD Player</td>
<td>25-FEB-00</td>
</tr>
<tr>
<td>4</td>
<td>Sony CD Player</td>
<td>25-FEB-00</td>
</tr>
<tr>
<td>6</td>
<td>Tascam CD Burner</td>
<td>25-FEB-00</td>
</tr>
<tr>
<td>2</td>
<td>Sony CD Player</td>
<td>24-FEB-00</td>
</tr>
<tr>
<td>5</td>
<td>Bose Speaker</td>
<td>22-FEB-00</td>
</tr>
<tr>
<td>1</td>
<td>Sony CD Player</td>
<td>20-FEB-00</td>
</tr>
</tbody>
</table>

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

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

**CONTEXT Query Grammar Examples**

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

**PROMPT**

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

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

PROMPT
PROMPT theme (ABOUT) query
PROMPT query: about(California)
select pk||' ==> '||text from test
where catsearch(text,
  '<query>
    <textquery grammar="context">
      about(California)
    </textquery>
    <score datatype="integer"/>
  </query>','')>0
order by pk;

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

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

insert into BOOKS values(1, '<author>NOAM CHOMSKY</author><subject>CIVIL RIGHTS</subject><language>ENGLISH</language><publisher>MIT
insert into BOOKS values(2, '<author>NICANOR PARRA</author><subject>POEMS AND ANTIPOEMS</subject><language>SPANISH</language><publisher>VASQUEZ</publisher>', '01-JAN-2001');

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

commit;

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

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

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

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

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

Syntax for CTXCAT Indextype in this chapter.

Oracle Text Application Developer’s Guide
CONTAINS

Use the **CONTAINS** operator in the \texttt{WHERE} clause of a \texttt{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 \texttt{SCORE} operator.

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

**Syntax**

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

**[schema.]column**

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

**text_query**

Specify one of the following:

- the query expression that defines your search in \texttt{column}.
- a marked-up document that specifies a query template. You can use one of the following templates:

**Query Rewrite Template**

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

```sql
{cat} {dog}
{cat} ; {dog}
{cat} AND {dog}
{cat} ACCUM {dog}
```
These queries are submitted as one query and results are returned with no duplication. In this example, the query returns documents that contain the phrase \textit{cat dog} as well as documents in which \textit{cat} is near \textit{dog}, and documents that have \textit{cat} and \textit{dog}.

This is done with the following template:

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

The operator \textsc{transform} is used to specify the rewrite rules and has the following syntax (note that it uses double parentheses):

\textsc{transform}((terms, prefix, suffix, connector))

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>terms</td>
<td>Specify the type of terms to be produced from the original query. You can specify either \textsc{TOKENS} or \textsc{THEMES}</td>
</tr>
<tr>
<td>prefix</td>
<td>Specify the literal string to be prepended to all the terms</td>
</tr>
<tr>
<td>suffix</td>
<td>Specify the literal string to be appended to all the terms.</td>
</tr>
<tr>
<td>connector</td>
<td>Specify the literal string to connect all the terms after applying prefix and suffix.</td>
</tr>
</tbody>
</table>

\textbf{Query Relaxation Template}

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

For example, the query of \textit{black pen} can be progressively relaxed to:

\begin{verbatim}
black pen
black NEAR pen
\end{verbatim}
black AND pen
black ACCUM pen

This is done with the following template

<query>
  <textquery lang="ENGLISH" grammar="CONTEXT">
    black pen
  </textquery>
  <progression>
    <seq>black pen</seq>
    <seq>black NEAR pen</seq>
    <seq>black AND pen</seq>
    <seq>black ACCUM pen</seq>
  </progression>
  <score datatype="INTEGER" algorithm="COUNT"/>
</query>

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

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

The phrase ‘dog cat mouse’ is interpreted as a phrase in CONTAINS. However, with CATSEARCH this is equivalent to a AND query of ‘dog AND cat AND mouse’. To specify that CONTAINS use the alternate grammar, we can issue the following template:

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

Alternate Language Template
Use this template to specify an alternate language.

<query><textquery lang="french">bon soir</textquery></query>
Alternate Scoring Template

Use this template to specify an alternate scoring algorithm. The following example specifies that the query use the CONTEXT grammar and return integer scores using the COUNT algorithm. This algorithm return score as number of query occurrences in document.

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

Template Attribute Values

The following table gives the possible values for template attributes:

<table>
<thead>
<tr>
<th>Tag Attribute</th>
<th>Description</th>
<th>Possible Values</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>grammar=</td>
<td>Specify the grammar of the query.</td>
<td>CONTEXT, CTXCAT</td>
<td>Returns score as integer between 0 and 100.</td>
</tr>
<tr>
<td>datatype=</td>
<td>Specify the type of number returned as score.</td>
<td>INTEGER, FLOAT</td>
<td>Returns score as its high precision floating point number between 0 and 100.</td>
</tr>
<tr>
<td>algorithm=</td>
<td>Specify the scoring algorithm to use.</td>
<td>DEFAULT, COUNT</td>
<td>Default. Returns scores as the number of occurrences in document.</td>
</tr>
<tr>
<td>lang=</td>
<td>Specify the language name.</td>
<td>ENGLISH, FRENCH, GERMAN, DUTCH</td>
<td></td>
</tr>
</tbody>
</table>

Template Grammar Definition

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

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

All tags and attributes values are case-sensitive.

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

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

Returns

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

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

Example

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

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

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

When the SCORE operator is called (for example, in a SELECT clause), the CONTAINS clause must reference the score label value as in the following example:
SELECT SCORE(1), title from newsindex
    WHERE CONTAINS(text, 'oracle', 1) > 0 ORDER BY SCORE(1) DESC;

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

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

**Grammar Template Example**

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

PROMPT create context and ctxcat indexes both with theme indexing on
PROMPT
create index tdrbqcq101x on test(text) indextype is ctxsys.context
parameters ('lexer theme_lexer');
create index tdrbqcq101cx on test(text) indextype is ctxsys.ctxcat
parameters ('lexer theme_lexer');

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

PROMPT *****  San Diego             ***********
PROMPT *****  CTXCAT grammar ***********
PROMPT ** should be interpreted as AND query ***
select pk||' ==> '||text from test
where contains(text,
    '<query>
    <textquery grammar="CTXCAT">San Diego</textquery>
    <score datatype="integer"/>
    </query>' ) > 0
order by pk;
PROMPT ***** Hitlist from CTXCAT index **************
select pk||' ==> '|text from test
where catsearch(text,'San Diego','')>0
order by pk;

Query Relaxation Template Example
The following query template defines a query relaxation sequence. The query of
black pen is issued in sequence as black pen then black NEAR pen then black AND pen
then black ACCUM pen. Query hits are returned in this sequence with no duplication
as long as the application needs results.
select id from docs where CONTAINS (text, 'black pen')>0;

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

Query Rewrite Example
The following template defines a query rewrite sequence. The query of kukui nut is
rewritten as follows:
{kukui} {nut}
{kukui};{nut}
{kukui} AND {nut}
{kukui} ACCUM {nut}
select id from docs where CONTAINS (text, 'kukui nut')>0;

<query>
  <textquery lang="ENGLISH" grammar="CONTEXT">kukui nut</textquery>
  <progression>
    <seq><rewrite>transform((TOKENS, "{", "}", ")")</rewrite></seq>
  </progression>
</query>
<seq><rewrite>transform((TOKENS, ",", ",\;\))</rewrite></seq>
<seq><rewrite>transform((TOKENS, ",", ",\;\\AND\))</rewrite></seq>
<seq><rewrite>transform((TOKENS, ",", ",\;\\ACCUM\))</rewrite></seq>
</progression>
</textquery>
<score datatype="INTEGER" algorithm="COUNT"/>
</query>'})>0;

Notes

**Querying Multi-Language Tables**
With the multi-lexer preference, you can create indexes from multi-language tables.

At query time, the multi-lexer examines the session’s language setting and uses the sub-lexer preference for that language to parse the query. If the language setting is not mapped, then the default lexer is used.

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

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

**Query Performance Limitation with a Partitioned Index**
Oracle Text supports the CONTEXT indexing and querying of a partitioned text table.

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

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

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

**Related Topics**

* Syntax for CONTEXT Indextype in this chapter
* Chapter 3, "CONTAINS Query Operators"
Oracle Text Application Developer's Guide

SCORE
CREATE INDEX

Note: This section describes the CREATE INDEX statement as it pertains to creating a Text domain index.
For a complete description of the CREATE INDEX statement, see Oracle Database SQL Reference.

Purpose

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

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

You can create the following types of Oracle Text indexes:

CONTEXT

This is an index on a text column. You query this index with the CONTAINS operator in the WHERE clause of a SELECT statement. This index requires manual synchronization after DML. See Syntax for CONTEXT Indextype.

CTXCAT

This is a combined index on a text column and one or more other columns. You query this index with the CATSEARCH operator in the WHERE clause of a SELECT statement. This type of index is optimized for mixed queries. This index is transactional, automatically updating itself with DML to the base table. See Syntax for CTXCAT Indextype.

CTXRULE

This is an index on a column containing a set of queries. You query this index with the MATCHES operator in the WHERE clause of a SELECT statement. See Syntax for CTXRULE Indextype.
CREATE INDEX

**CTXPATH**

Create this index when you need to speed up existsNode() queries on an XMLType column. See Syntax for CTXPATH Indextype.

**Required Privileges**

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

**Syntax for CONTEXT Indextype**

Use this indextype to create an index on a text column. You query this index with the `CONTAINS` operator in the `WHERE` clause of a `SELECT` statement. This index requires manual synchronization after DML.

```
CREATE INDEX [schema.]index ON [schema.]table(column) INDEXTYPE IS ctxsys.context [ONLINE]
[LOCAL [(PARTITION [partition] [PARAMETERS('paramstring')])
[, PARTITION [partition] [PARAMETERS('paramstring')]])]
[PARAMETERS(paramstring)] [PARALLEL n] [UNUSABLE];
```

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

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

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

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

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

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.

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

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

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

Note: With the CTXCAT index type, you can create indexes on text and structured columns. See Syntax for CTXCAT Indextype in this chapter.

ONLINE
Creates the index while enabling inserts/updates/deletes (DML) on the base table.

During indexing, Oracle Text enqueues DML requests in a pending queue. At the end of the index creation, Oracle Text locks the base table. During this time DML is blocked.

Limitations
The following limitations apply to using ONLINE:
- At the very beginning or very end of this process, DML might fail.
- Local partition index online creation not supported with ONLINE.
- ONLINE is supported for CONTEXT indexes only
- ONLINE cannot be used with PARALLEL

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

You can specify the list of index partition names with partition. If you do not specify a partition name, the system assigns one. The order of the index partition list must correspond to the table partition by order.
The PARAMETERS clause associated with each partition specifies the parameters string specific to that partition. You can only specify `sync (manual | every | on commit), memory and storage` for each index partition.

You can query the views `CTX_INDEX_PARTITIONS` or `CTX_USER_INDEX_PARTITIONS` to find out index partition information, such as index partition name, and index partition status.

You cannot use the ONLINE parameter with this operation.

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

**Query Performance Limitation with Partitioned Index**

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

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

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

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

Parallel indexing is supported for creating a local partitioned index.

**See Also:**  
"Parallel Indexing”  
"Creating a Local Partitioned Index in Parallel”

Performance Tuning chapter in *Oracle Text Application Developer’s Guide*

**Performance**

Parallel indexing can speed up indexing when you have large amounts of data to index and when your operating system supports multiple CPUs.
**Note:** Using PARALLEL to create a local partitioned index enables parallel queries. (Creating a non-partitioned index in parallel does not turn on parallel query processing.)

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

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

**Limitations**
The following limitations apply to using PARALLEL:

- Parallel indexing is supported only for CONTEXT index
- PARALLEL cannot be used with ONLINE.

**UNUSABLE**
Create an unusable index. This creates index metadata only and exits immediately.

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

**See Also:** "Creating a Local Partitioned Index in Parallel"

**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:

```sql
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]
```

SQL Statements and Operators 1-47
CREATE INDEX

[STOPLIST stoplist]
[SECTION GROUP section_group]
[MEMORY memsize]
[POPULATE | NOPOPULATE]
[[METADATA] SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)]
[TRANSACTIONAL]

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

---

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

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

---

**DATASTORE datastore_pref**

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

**FILTER filter_pref**

Specify the name of your filter preference. Use the filter preference to specify how to filter formatted documents to plain text or HTML. See Filter Types in Chapter 2, "Oracle Text Indexing Elements".

**CHARSET COLUMN charset_column_name**

Specify the name of the character set column. This column must be in the same table as the text column, and it must be of type CHAR, VARCHAR, or VARCHAR2. Use this column to specify the document character set for conversion to the database character set. The value is case insensitive. You must specify a Globalization Support character set string such as JA16EUC.

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

For all rows containing the keywords 'AUTO' or 'AUTOMATIC', Oracle Text will apply statistical techniques to determine the character set of the documents and modify document indexing appropriately.

You use this column when you have plain text or HTML documents with different character sets or in a character set different from the database character set.
Note: Documents are not marked for re-indexing when only the
charset column changes. The indexed column must be updated to
flag the re-index.

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

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

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

- **TEXT**
- **BINARY**
- **IGNORE**

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

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

**IGNORE** indicates that the row is to be ignored during indexing. Use this value
when you need to bypass rows that contain data incompatible with text indexing
such as image data.

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

**LEXER lexer_pref**
Specify the name of your lexer or multi-lexer preference. Use the lexer preference to
identify the language of your text and how text is tokenized for indexing. See **Lexer Types** in Chapter 2, "Oracle Text Indexing Elements".
**LANGUAGE COLUMN language_column_name**
Specify the name of the language column when using a multi-lexer preference. See **MULTI_LEXER** in Chapter 2, "Oracle Text Indexing Elements".

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.

For all rows containing the keywords 'AUTO' or 'AUTOMATIC', Oracle Text will apply statistical techniques to determine the language of the documents and modify document indexing appropriately.

---

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

---

**WORDLIST wordlist_pref**
Specify the name of your wordlist preference. Use the wordlist preference to enable features such as fuzzy, stemming, and prefix indexing for better wildcard searching. See **Wordlist Type** in Chapter 2, "Oracle Text Indexing Elements".

**STORAGE storage_pref**
Specify the name of your storage preference for the Text index. Use the storage preference to specify how the index tables are stored. See **Storage Types** in Chapter 2, "Oracle Text Indexing Elements".

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

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

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

\[
\text{memsize} = \text{number}\{K|M|G\}
\]

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

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

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

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

**POPULATE | NOPOPULATE**

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

---

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

This option is not valid with CTXXPATH indexes.

---

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

**[METADATA] SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)**

Specify `SYNC` for automatic synchronization of the `CONTEXT` index when there are inserts, updates or deletes to the base table. You can specify one of the following `SYNC` methods:

<table>
<thead>
<tr>
<th>SYNC type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MANUAL</td>
<td>No automatic synchronization. This is the default. You must manually synchronize the index with <code>CTX_DDL.SYNC_INDEX</code>.</td>
</tr>
</tbody>
</table>
CREATE INDEX

<table>
<thead>
<tr>
<th>SYNC type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVERY &quot;interval-string&quot;</td>
<td>Automatically synchronize the index at a regular interval specified by the value of interval-string. interval-string takes the same syntax as that for scheduler jobs. Automatic synchronization using EVERY requires that the index creator have CREATE JOB privileges. Make sure that interval-string is set to a long enough period that any previous sync jobs will have completed; otherwise, the sync job may hang. interval-string must be enclosed in double quotes, and any single quote within interval-string must be escaped with another single quote. See Enabling Automatic Index Synchronization on page 1-54 for an example of automatic sync syntax.</td>
</tr>
<tr>
<td>ON COMMIT</td>
<td>Synchronize the index immediately after a commit. The commit does not return until the sync is complete. (Since the synchronization is performed as a separate transaction, there may be a period, usually small, when the data is committed but index changes are not.) The operation uses the memory specified with the memory parameter. Note that the sync operation has its own transaction context. If this operation fails, the data transaction still commits. Index synchronization errors are logged in the CTX_USER_INDEX_ERRORS view. See Viewing Index Errors under CREATE INDEX. See Enabling Automatic Index Synchronization on page 1-54 for an example of ON COMMIT syntax.</td>
</tr>
</tbody>
</table>

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

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

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

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

See the Oracle Database Administrator’s Guide for information on job scheduling.

TRANSACTIONAL

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

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

You must still synchronize your text indexes from time to time (with CTX_DDL_SYNC_INDEX) to bring pending rowids into the index. Query performance degrades as the number of unsynchronized rowids increases. For that reason, Oracle recommends setting up your index to use automatic synchronization with the EVERY parameter. (See [METADATA] SYNC (MANUAL | EVERY "interval-string" | ON COMMIT) on page 1-51.)

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

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

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

**CREATE INDEX: CONTEXT Index Examples**

The following sections give examples of creating a CONTEXT index.

**Creating CONTEXT Index Using Default Preferences**

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

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

**See Also:** For more information about default settings, see "Default Index Parameters" in Chapter 2.

Also refer to Oracle Text Application Developer's Guide.
Creating CONTEXT Index with Custom Preferences

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

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

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

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

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

Enabling Automatic Index Synchronization

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

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

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

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

CREATE INDEX tdrm02x ON tdrm02i02x(text)
  INDEXTYPE IS CTXSYS.CONTEXT local
  (PARTITION tdrm02x_i1 PARAMETERS('MEMORY 20m SYNC(ON COMMIT)'));
```

1-54 Oracle Text Reference
PARTITION tdrm02x_i3) PARAMETERS('SYNC (EVERY "NEXT_DAY(TRUNC(SYSDATE), '"MONDAY"') + 15/24")');

See the Oracle Database Administrator’s Guide for information on job scheduling syntax.

Creating CONTEXT Index with Multi-Lexer Preference

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

```sql
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:

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

See Also: For more information about creating multi-lexer preferences, see MULTI_LEXER in Chapter 2.

Creating a Local Partitioned Index

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

```sql
PROMPT create partitioned table and populate it

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

PROMPT create partitioned index

CREATE INDEX part_idx on part Tab(b) INDEXTYPE IS CTXSYS.CONTEXT LOCAL (partition p_idx1, partition p_idx2, partition p_idx3);
```
Parallel Indexing

Parallel indexing can improve index performance when you have multiple CPUs. To create an index in parallel, use the PARALLEL clause with a parallel degree. This example uses a parallel degree of 3:

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

Creating a Local Partitioned Index in Parallel

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

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

Parallelism with CREATE INDEX

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

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

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

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

create index part_tab3x on part_tab3(text)
```
CREATE INDEX

indextype is ctxsys.context local (partition part_tabx1,
     partition part_tabx2,
     partition part_tabx3)
parallel 2;

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

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

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

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

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

exec dbms_pclxutil.build_part_index(jobs_per_batch=>3,
  procs_per_job=>2,
  tab_name=>'PART_TAB3',
  idx_name=>'PART_TAB3X',
  force_opt=>TRUE);
Viewing Index Errors
After a CREATE INDEX or ALTER INDEX operation, you can view index errors with Oracle Text views. To view errors on your indexes, query the CTX_USER_INDEX_ERRORS view. To view errors on all indexes as CTXSYS, query the CTX_INDEX_ERRORS view.

For example, to view the most recent errors on your indexes, you can issue:

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

Deleting Index Errors
To clear the index error view, you can issue:

```
DELETE FROM ctx_user_index_errors;
```

Syntax for CTXCAT Indextype
The CTXCAT index is a combined index on a text column and one or more other columns. You query this index with the CATSEARCH operator in the WHERE clause of a SELECT statement. This type of index is optimized for mixed queries. This index is transactional, automatically updating itself with DML to the base table.

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

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

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

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

- The VPD policy is created such that it does not apply to INDEX statement type, which is the default
The policy function returns a null predicate for the current user.

- The user (index owner) is SYS.
- The user has the EXEMPT ACCESS POLICY privilege.

**Supported Preferences**

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

You must always ensure that your columns have non-NULL values before and after indexing.

See “Creating a CTXCAT Index” on page 1-60.

**Index Performance and Size Considerations**
Although a CTXCAT index offers query performance benefits, creating the index has its costs. The time Oracle Text takes to create a CTXCAT index depends on its total size, and the total size of a CTXCAT index is directly related to

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

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

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

**See Also:** Oracle Text Application Developer’s Guide for more information about creating CTXCAT indexes and its benefits.

**Other Preferences**
When you create an index of type CTXCAT, you can use the following supported index preferences in the parameters string:
Table 1–1  Supported CTXCAT Index Preferences

<table>
<thead>
<tr>
<th>Preference Class</th>
<th>Supported Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datastore</td>
<td>This preference class is not supported for CTXCAT.</td>
</tr>
<tr>
<td>Filter</td>
<td>This preference class is not supported for CTXCAT.</td>
</tr>
<tr>
<td>Lexer</td>
<td>BASIC_Lexer (index_themes attribute not supported)</td>
</tr>
<tr>
<td></td>
<td>CHINESE_Lexer</td>
</tr>
<tr>
<td></td>
<td>CHINESE_VGRAM_Lexer</td>
</tr>
<tr>
<td></td>
<td>JAPANESE_Lexer</td>
</tr>
<tr>
<td></td>
<td>JAPANESE_VGRAM_Lexer</td>
</tr>
<tr>
<td></td>
<td>KOOREAN_Lexer</td>
</tr>
<tr>
<td></td>
<td>KOOREAN_Lexer</td>
</tr>
<tr>
<td>Wordlist</td>
<td>BASIC_WORDLIST</td>
</tr>
<tr>
<td>Storage</td>
<td>BASIC_STORAGE</td>
</tr>
<tr>
<td>Stoplist</td>
<td>Supports single language stoplists only (BASIC_STOPLIST type.)</td>
</tr>
<tr>
<td>Section Group</td>
<td>This preference class is not supported for CTXCAT.</td>
</tr>
</tbody>
</table>

Unsupported Preferences and Parameters

When you create a CTXCAT index, you cannot specify datastore, filter and section group preferences. You also cannot specify language, format, and charset columns as with a CONTEXT index.

Creating a CTXCAT Index

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

Consider a table called AUCTION with the following schema:

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

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

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

To create the indexes, first create the index set preference then add the required indexes to it:

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

Create the CTXCAT index with CREATE INDEX as follows:

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

**Querying a CTXCAT Index**

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

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

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

**Syntax for CTXRULE Indextype**

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

```sql
CREATE INDEX [schema.]index on [schema.]table(rule_col) INDEXTYPE IS
ctxsys.ctrxrule
[PARAMETERS ('[lexer lexer_pref] [storage storage_pref]
[section group section_pref] [wordlist wordlist_pref]
[classifier classifier_pref]');
[PARALLEL n]);
```
[schema.]table(column)
Specify the name of the table and rule column to index. The rules can be query
compatible strings, query template strings, or binary support vector machine rules.

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

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

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

lexer_pref
Specify the lexer preference to be used for processing the queries and the
documents to be classified with the MATCHES function. Currently, the BASIC_LEXER,
CHINESE_LEXER, JAPANESE_LEXER, and KOREAN_LEXER lexer types
are supported.

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

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

storage_pref
Specify the storage preference for the index on the queries. Use the storage
preference to specify how the index tables are stored. See Storage Types in
Chapter 2, "Oracle Text Indexing Elements".

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

- BASIC_SECTION_GROUP
- HTML_SECTION_GROUP
XML_SECTION_GROUP
AUTO_SECTION_GROUP

See Section Group Types in Chapter 2, "Oracle Text Indexing Elements".

CTXRULE does not support special sections.

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

classifier_pref
Specify the classifier preference. See Classifier Types in Chapter 2, "Oracle Text Indexing Elements". You must use the same preference name you specify with CTX_CLS.TRAIN.

Example for Creating a CTXRULE Index
See the Oracle Text Application Developer’s Guide for a complete example of using the CTXRULE indextype in a document routing application.

Syntax for CTXXPATH Indextype
Create this index when you need to speed up existsNode() queries on an XMLType column.

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

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

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

storage_pref
Specify the storage preference for the index on the queries. Use the storage preference to specify how the index tables are stored. See Storage Types in Chapter 2, "Oracle Text Indexing Elements".

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

memsize = number[M|G|K]

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

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

The default is the value specified for DEFAULT_INDEX_MEMORY in CTX_PARAMETERS.

CTXXPATH Examples

Index creation on an XMLType column:

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

or

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

Querying the table with existsNode:

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

See Also: Oracle XML DB Developer’s Guide for information on using the CTXXPATH indextype.

Related Topics

CTX_DDL.CREATE PREFERENCE in Chapter 7, "CTX_DDL Package".
CTX_DDL.CREATE_STOPLIST in Chapter 7, "CTX_DDL Package".
CTX_DDL.CREATE_SECTION_GROUP in Chapter 7, "CTX_DDL Package".
ALTER INDEX
CATSEARCH
**DROP INDEX**

**Note:** This section describes the DROP INDEX statement as it pertains to dropping a Text domain index.

For a complete description of the DROP INDEX statement, see Oracle Database SQL Reference.

**Purpose**

Use DROP INDEX to drop a specified Text index.

**Syntax**

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

**[force]**

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

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

**Examples**

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

```sql
DROP INDEX doc_index;
```

**Related Topics**

- ALTER INDEX
- CREATE INDEX
MATCHES

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

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

MATCHES returns a number in the range of 0 to 100. Zero means no match. When this number is greater than zero, there are one or more matches. You can use the label parameter and MATCH_SCORE to obtain this number.

Limitation

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

Syntax

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

column
Specify the column containing the indexed query set.

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

label
Optionally specify the label that identifies the score generated by the MATCHES operator. You use this label with MATCH_SCORE.

Matches Example

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

The example issues the MATCHES query with a document string to be classified. The SELECT statement returns all rows (queries) that are satisfied by the document:
create table querytable (classification varchar2(64), text varchar2(4000));
insert into querytable values ('common names', 'smith OR jones OR brown');
insert into querytable values ('countries', 'United States OR Great Britain OR France');
insert into querytable values ('Oracle DB', 'oracle NEAR database');

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

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

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

Simple Classification Examples

The MATCHES operator is used in simple and supervised classification. For more extended examples, see the "Building Classification Applications" chapter in the Oracle Text Application Developer's Guide.

Related Topics

Syntax for CTXRULE Indextype in this chapter.

CTX_CLS.TRAIN

Oracle Text Application Developer’s Guide
MATCH_SCORE

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

This operator returns a score in the range 0 to 100. You can use the matching score to apply a category specific threshold to a particular category.

Syntax

MATCH_SCORE(label NUMBER)

label
Specify a number to identify the score produced by the query. You use this number to identify the MATCHES clause which returns this score.

Example

To get the matching score, use

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

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

Syntax

`SCORE(label NUMBER)`

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

Example

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

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

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

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

Related Topics

CONTAINS

Appendix F, “Scoring Algorithm”
This chapter describes the various elements you can use to create your Oracle Text index.

The following topics are discussed in this chapter:

- **Overview**
- **Datastore Types**
- **Filter Types**
- **Lexer Types**
- **Wordlist Type**
- **Storage Types**
- **Section Group Types**
- **Classifier Types**
- **Cluster Types**
- **Stoplists**
- **System-Defined Preferences**
- **System Parameters**

**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 Text indexes text:
This chapter describes how to set each preference. You enable an option by creating a preference with one of the types described in this chapter.

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

### Creating Preferences

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

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

You specify indexing preferences with `CREATE INDEX` and `ALTER INDEX`; indexing preferences determine how your index is created. For example, lexer preferences indicate the language of the text to be indexed. You can create and specify your own (user-defined) preferences or you can utilize system-defined preferences.
To create a stoplist, use `CTX_DDL.CREATE_STOPLIST`. You can add stopwords to a stoplist with `CTX_DDL.ADD_STOPWORD`.

To create section groups, use `CTX_DDL.CREATE_SECTION_GROUP` and specify a section group type. You can add sections to section groups with `CTX_DDL.ADD_ZONE_SECTION` or `CTX_DDL.ADD_FIELD_SECTION`.

**Datastore Types**

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

<table>
<thead>
<tr>
<th>Datastore Type</th>
<th>Use When</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIRECT_DATASTORE</td>
<td>Data is stored internally in the text column. Each row is indexed as a single document.</td>
</tr>
<tr>
<td>MULTI_COLUMN_DATASTORE</td>
<td>Data is stored in a text table in more than one column. Columns are concatenated to create a virtual document, one for each row.</td>
</tr>
<tr>
<td>DETAIL_DATASTORE</td>
<td>Data is stored internally in the text column. Document consists of one or more rows stored in a text column in a detail table, with header information stored in a master table.</td>
</tr>
<tr>
<td>FILE_DATASTORE</td>
<td>Data is stored externally in operating system files. Filenames are stored in the text column, one for each row.</td>
</tr>
<tr>
<td>NESTED_DATASTORE</td>
<td>Data is stored in a nested table.</td>
</tr>
<tr>
<td>URL_DATASTORE</td>
<td>Data is stored externally in files located on an intranet or the Internet. Uniform Resource Locators (URLs) are stored in the text column.</td>
</tr>
<tr>
<td>USER_DATASTORE</td>
<td>Documents are synthesized at index time by a user-defined stored procedure.</td>
</tr>
</tbody>
</table>

**DIRECT_DATASTORE**

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

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

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

```sql
create table mytable(id number primary key, docs clob);
insert into mytable values(111555,'this text will be indexed');
insert into mytable values(111556,'this is a direct_datastore example');
commit;
create index myindex on mytable(docs)
    indextype is ctxsys.context
    parameters ('DATASTORE CTXSYS.DEFAULT_DATASTORE');
```

MULTI_COLUMN_DATASTORE

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

MULTI_COLUMN_DATASTORE has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>columns</td>
<td>Specify a comma separated list of columns to be concatenated during indexing. You can also specify any expression allowable for the select statement column list for the base table. This includes expressions, PL/SQL functions, column aliases, and so on. NUMBER and DATE column types are supported. They are converted to text before indexing using the default format mask. The TO_CHAR function can be used in the column list for formatting. RAW and BLOB columns are directly concatenated as binary data. LONG, LONG RAW, NCHAR, and NCLOB, nested table columns and collections are not supported. The column list is limited to 500 bytes.</td>
</tr>
</tbody>
</table>
Indexing and DML

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

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

**MULTI_COLUMN_DATASTORE Example**

The following example creates a multi-column datastore preference called *my_multi* with three text columns:

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

**MULTI_COLUMN_DATASTORE Filter Example**

The following example creates a multi-column datastore preference and denotes that the *bar* column is to be filtered with the INSO_FILTER.

```sql
ctx_ddl.create_preference('MY_MULTI', 'MULTI_COLUMN_DATASTORE');
ctx_ddl.set_attribute('MY_MULTI', 'COLUMNS', 'foo,bar');
ctx_ddl.set_attribute('MY_MULTI', 'FILTER', 'N,Y');
```
The multi-column datastore fetches the content of the foo and bar columns, filters bar, then composes the compound document as:

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

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

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

This filters only the column zoo.

**Tagging Behavior**

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

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

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

This produces the following virtual text for indexing:

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

The system indexes the text between the tags, ignoring the tags themselves.
Indexing Columns as Sections
To index these tags as sections, you can optionally create field sections with the BASIC_SECTION_GROUP.

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

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

For example, if your expression is as follows:

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

then it produces the following virtual text:

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

If your expression is as follows:

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

then it produces the following virtual text:

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

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

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

produces the following virtual text:

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

For lowercase tags, use the following:

```exec ctx_ddl.set_attribute('mymds', 'COLUMNS', 'foo "foo"');
```
This expression produces:

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

**DETAIL_DATASTORE**

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

`DETAIL_DATASTORE` has the following attributes:

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

**Synchronizing Master/Detail Indexes**

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

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

**Example Master/Detail Tables**

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

**Master Table Example**

Master tables define the documents in a master/detail relationship. You assign an identifying number to each document. The following table is an example master table, called `my_master`:
Detail Table Example  
Detail tables contain the text for a document, whose content is usually stored across a number of rows. The following detail table `my_detail` is related to the master table `my_master` with the `article_id` column. This column identifies the master document to which each detail row (sub-document) belongs.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>article_id</td>
<td>NUMBER</td>
<td>Document ID, unique for each document (Primary Key)</td>
</tr>
<tr>
<td>author</td>
<td>VARCHAR2(30)</td>
<td>Author of document</td>
</tr>
<tr>
<td>title</td>
<td>VARCHAR2(50)</td>
<td>Title of document</td>
</tr>
<tr>
<td>body</td>
<td>CHAR(1)</td>
<td>Dummy column to specify in CREATE INDEX</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>TRUE</td>
</tr>
<tr>
<td>detail_table</td>
<td>my_detail</td>
</tr>
<tr>
<td>detail_key</td>
<td>article_id</td>
</tr>
<tr>
<td>detail_lineno</td>
<td>seq</td>
</tr>
<tr>
<td>detail_text</td>
<td>text</td>
</tr>
</tbody>
</table>
You use CTX_DDL.CREATE_PREFERENCE to create a preference with DETAIL_DATASTORE. You use CTX_DDL.SET_ATTRIBUTE to set the attributes for this preference as described earlier. The following example shows how this is done:

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

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

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

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

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

**FILE_DATASTORE**

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

---

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

---

FILE_DATASTORE has the following attribute(s):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>path1:path2:pathn</td>
</tr>
</tbody>
</table>
### path
Specify the full directory path name of the files stored externally in a file system. When you specify the full directory path as such, you need only include file names in your text column.

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

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

### PATH Attribute Limitations
The PATH attribute has the following limitations:

- If you specify a PATH attribute, you can only use a simple filename in the indexed column. You cannot combine the PATH attribute with a path as part of the filename. If the files exist in multiple folders or directories, you must leave the PATH attribute unset, and include the full file name, with PATH, in the indexed column.

- On Windows systems, the files must be located on a local drive. They cannot be on a remote drive, whether the remote drive is mapped to a local drive letter.

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

```sql
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.

```sql
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:

```sql
create index myindex on mytable(docs)
```
URL_DATASTORE

Use the URL_DATASTORE type for text stored:

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

You store each URL in a single text field.

URL Syntax

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

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

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

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

As this syntax is partially compliant with the RFC 1738 specification, the following restriction holds for the URL syntax:

- The URL must contain only printable ASCII characters. 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 supported only for the ftp access scheme.

---

URL_DATASTORE Attributes

URL_DATASTORE has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>Specify the timeout in seconds. The valid range is 15 to 3600 seconds. The default is 30.</td>
</tr>
</tbody>
</table>
### Attribute Values

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxthreads</td>
<td>Specify the maximum number of threads that can be running simultaneously. Use a number between 1 and 1024. The default is 8.</td>
</tr>
<tr>
<td>urlsizel</td>
<td>Specify the maximum length of URL string in bytes. Use a number between 32 and 65535. The default is 256.</td>
</tr>
<tr>
<td>maxurls</td>
<td>Specify maximum size of URL buffer. Use a number between 32 and 65535. The defaults is 256.</td>
</tr>
<tr>
<td>maxdocsize</td>
<td>Specify the maximum document size. Use a number between 256 and 2,147,483,647 bytes (2 gigabytes). The defaults is 2,000,000.</td>
</tr>
<tr>
<td>http_proxy</td>
<td>Specify the host name of http proxy server. Optionally specify port number with a colon in the form hostname:port.</td>
</tr>
<tr>
<td>ftp_proxy</td>
<td>Specify the host name of ftp proxy server. Optionally specify port number with a colon in the form hostname:port.</td>
</tr>
<tr>
<td>no_proxy</td>
<td>Specify the domain for no proxy server. Use a comma separated string of up to 16 domain names.</td>
</tr>
</tbody>
</table>

### 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.

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

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

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

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

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

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

**no_proxy**
Specify a string of domains (up to sixteen, separate by commas) which are found in most, if not all, of the machines in your intranet. When one of the domains is encountered in a host name, no request is sent to the machine(s) specified for ftp_proxy and http_proxy. Instead, the request is processed directly by the host machine identified in the URL.
For example, if the string \texttt{us.oracle.com}, \texttt{uk.oracle.com} is entered for \texttt{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 \texttt{URL\_PREF} for which the \texttt{http\_proxy}, \texttt{no\_proxy}, and \texttt{timeout} attributes are set. The defaults are used for the attributes that are not set.

```sql
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:

```sql
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 \texttt{URL\_PREF} as the datastore:

```sql
create index datastores_text on urls ( docs )
indextype is ctxsys.context
parameters ( 'Datastore URL\_PREF' );
```

**USER\_DATASTORE**

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

The \texttt{USER\_DATASTORE} has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure</td>
<td>Specify the procedure that synthesizes the document to be indexed. This procedure can be owned by any user and must be executable by the index owner.</td>
</tr>
</tbody>
</table>
procedure
Specify the name of the procedure that synthesizes the document to be indexed. This specification must be in the form PROCEDURENAME or PACKAGENAME.PROCEDURENAME. You can also specify the schema owner name.

The procedure you specify must have two arguments defined as follows:

```sql
procedure (r IN ROWID, c IN OUT NOCOPY <output_type>)
```

The first argument `r` must be of type ROWID. The second argument `c` must be of type `output_type`. `NOCOPY` is a compiler hint that instructs Oracle Text to pass parameter `c` by reference if possible.

**Note:** The procedure name and its arguments can be named anything. The arguments `r` and `c` are used in this example for simplicity.

The stored procedure is called once for each row indexed. Given the rowid of the current row, procedure must write the text of the document into its second argument, whose type you specify with `output_type`.

**Constraints**
The following constraints apply to procedure:

- `procedure` can be owned by any user, but the user must have database permissions to execute `procedure` correctly
- `procedure` must be executable by the index owner
- `procedure` must not issue DDL or transaction control statements like COMMIT
Editing Procedure after Indexing

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

output_type

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

USER_DATASTORE with CLOB Example

Consider a table in which the author, title, and text fields are separate, as in the articles table defined as follows:

```sql
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:

```sql
create procedure myproc(rid in rowid, tlob in out clob nocopy) is
    begin
        for c1 in (select author, title, text from articles where rowid = rid)
            loop
                dbms_lob.writeappend(tlob, length(c1.title), c1.title);
                dbms_lob.writeappend(tlob, length(c1.author), c1.author);
                dbms_lob.writeappend(tlob, length(c1.text), c1.text);
            end loop;
    end;
```

This procedure takes in a rowid and a temporary CLOB locator, and concatenates all the article’s columns into the temporary CLOB. The for loop executes only once.

The user appowner creates the preference as follows:
begin
cxt_ddl.create_preference('myud', 'user_datastore');
cxt_ddl.set_attribute('myud', 'procedure', 'myproc');
cxt_ddl.set_attribute('myud', 'output_type', 'CLOB');
end;

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

**USER_DATASTORE with BLOB_LOC Example**
The following procedure might be used with OUTPUT_TYPE BLOB_LOC:

```sql
procedure myds(rid in rowid, dataout in out nocopy blob)
is
  l_dtype varchar2(10);
  l_pk    number;
begin
  select dtype, pk into l_dtype, l_pk from mytable where rowid = rid;
  if (l_dtype = 'MOVIE') then
    select movie_data into dataout from movietab where fk = l_pk;
  elsif (l_dtype = 'SOUND') then
    select sound_data into dataout from soundtab where fk = l_pk;
  end if;
end;
```

The user appowner creates the preference as follows:

begin
  ctx_ddl.create_preference('myud', 'user_datastore');
  ctx_ddl.set_attribute('myud', 'procedure', 'myproc');
  ctx_ddl.set_attribute('myud', 'output_type', 'blob_loc');
end;

**NESTED_DATASTORE**
Use the nested datastore type to index documents stored as rows in a nested table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nested_column</td>
<td>Specify the name of the nested table column. This attribute is required. Specify only the column name. Do not specify schema owner or containing table name.</td>
</tr>
</tbody>
</table>
When using the nested table datastore, you must index a dummy column, because the extensible indexing framework disallows indexing the nested table column. See the example.

DML on the nested table is not automatically propagated to the dummy column used for indexing. For DML on the nested table to be propagated to the dummy column, your application code or trigger must explicitly update the dummy column.

Filter defaults for the index are based on the type of the nested_text column.

During validation, Oracle Text checks that the type exists and that the attributes you specify for nested_lineno and nested_text exist in the nested table type. Oracle Text does not check that the named nested table column exists in the indexed table.

**NESTED_DATASTORE Example**

This section shows an example of using the NESTED_DATASTORE type to index documents stored as rows in a nested table.

**Create the Nested Table**  The following code creates a nested table and a storage table mytab for the nested table:

```sql
create type nt_rec as object (
    lno number, -- line number
    ltxt varchar2(80) -- text of line
);
```
create type nt_tab as table of nt_rec;
create table mytab (  
id number primary key, -- primary key  
dummy char(1), -- dummy column for indexing  
doc nt_tab -- nested table )
nested table doc store as myntab;

Insert Values into Nested Table  The following code inserts values into the nested table for the parent row with id equal to 1.
insert into mytab values (1, null, nt_tab());
insert into table (select doc from mytab where id=1) values (1, 'the dog');
insert into table (select doc from mytab where id=1) values (2, 'sat on mat ');
commit;

Create Nested Table Preferences  The following code sets the preferences and attributes for the NESTED_DATASTORE according to the definitions of the nested table type nt_tab and the parent table mytab:
begin  
-- create nested datastore pref
ctx_ddl.create_preference('ntds','nested_datastore');

-- nest tab column in main table
ctx_ddl.set_attribute('ntds','nested_column', 'doc');

-- nested table type
ctx_ddl.set_attribute('ntds','nested_type', 'scott.nt_tab');

-- lineno column in nested table
ctx_ddl.set_attribute('ntds','nested_lineno','lno');

--text column in nested table
ctx_ddl.set_attribute('ntds','nested_text', 'ltxt');
end;

Create Index on Nested Table  The following code creates the index using the nested table datastore:
create index myidx on mytab(dummy) -- index dummy column, not nest table indextype is ctxsys.context parameters ('datastore ntds');
Query Nested Datastore  The following select statement queries the index built from a nested table:

```
select * from mytab where contains(dummy, 'dog and mat')>0;
-- returns document 1, since it has dog in line 1 and mat in line 2.
```

**Filter Types**

Use the filter types to create preferences that determine how text is filtered for indexing. Filters allow word processor and formatted documents as well as plain text, HTML, and XML documents to be indexed.

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

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

<table>
<thead>
<tr>
<th>Filter Preference type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARSET_FILTER</td>
<td>Character set converting filter</td>
</tr>
<tr>
<td>INSO_FILTER</td>
<td>Inso filter for filtering formatted documents</td>
</tr>
<tr>
<td>NULL_FILTER</td>
<td>No filtering required. Use for indexing plain text, HTML, or XML documents</td>
</tr>
<tr>
<td>MAIL_FILTER</td>
<td>Use the MAIL_FILTER to transform RFC-822, RFC-2045 messages into indexable text.</td>
</tr>
<tr>
<td>USER_FILTER</td>
<td>User-defined external filter to be used for custom filtering</td>
</tr>
<tr>
<td>PROCEDURE_FILTER</td>
<td>User-defined stored procedure filter to be used for custom filtering</td>
</tr>
</tbody>
</table>

**CHARSET_FILTER**

Use the CHARSET_FILTER to convert documents from a non-database character set to the character set used by the database.

CHARSET_FILTER has the following attribute:
UTF-16 Big- and Little-Endian Detection

If your character set is UTF-16, you can specify UTF16AUTO to automatically detect big- or little-endian data. Oracle Text does so by examining the first two bytes of the document row.

If the first two bytes are 0xFE, 0xFF, the document is recognized as little-endian and the remainder of the document minus those two bytes is passed on for indexing.

If the first two bytes are 0xFF, 0xFE, the document is recognized as big-endian and the remainder of the document minus those two bytes is passed on for indexing.

If the first two bytes are anything else, the document is assumed to be big-endian and the whole document including the first two bytes is passed on for indexing.

Indexing Mixed-Character Set Columns

A mixed character set column is one that stores documents of different character sets. For example, a text table might store some documents in WE8ISO8859P1 and others in UTF8.

To index a table of documents in different character sets, you must create your base table with a character set column. In this column, you specify the document character set on a per-row basis. To index the documents, Oracle Text converts the documents into the database character set.

Character set conversion works with the CHARSET_FILTER. When the charset column is NULL or not recognized, Oracle Text assumes the source character set is the one specified in the charset attribute.

See Also: Oracle Database Globalization Support Guide for more information about the supported Globalization Support character sets.
Filter Types

Indexing Mixed-Character Set Example  For example, create the table with a charset column:

```sql
create table hdocs (  
id number primary key,  
fmt varchar2(10),  
cset varchar2(20),  
text varchar2(80)  
);  

Create a preference for this filter:

begin  
cxt_ddl.create.preference('cs_filter', 'CHARSET_FILTER');  
ctx_ddl.set_attribute('cs_filter', 'charset', 'UTF8');  
end

Insert plain-text documents and name the character set:

insert into hdocs values(1, 'text', 'WE8ISO8859P1', '/docs/iso.txt');  
insert into hdocs values (2, 'text', 'UTF8', '/docs/utf8.txt');  
commit;

Create the index and name the charset column:

create index hdocsx on hdocs(text) indextype is ctxsys.context  
parameters ('datastore ctxsys.file_datastore  
filter cs_filter  
format column fmt  
charset column cset');
```

**INSO_FILTER**

The INSO_FILTER is a universal filter that filters most document formats, including PDF, Microsoft Word™, and MacWrite II™ documents. This filtering technology, called Outside In HTML Export™ and Outside In Viewer Technology™, is licensed from Stellant Chicago, Inc.

Use it for indexing single-format and mixed-format columns.

**Note:** Character set conversion also works with the INSO_FILTER when the document format column is set to TEXT.
This filter automatically bypasses plain-text, HTML, and XML documents.

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

The INSO_FILTER has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>Specify the INSO_FILTER timeout in seconds. Use a number between 0 and 42,949,672. Default is 120. Setting this value 0 disables the feature. How this wait period is used depends on how you set timeout_type. This feature is disabled for rows for which the corresponding charset and format column cause the INSO_FILTER to bypass the row, such as when format is marked TEXT. Use this feature to prevent the Oracle Text indexing operation from waiting indefinitely on a hanging filter operation.</td>
</tr>
<tr>
<td>timeout_type</td>
<td>Specify either HEURISTIC or FIXED. Default is HEURISTIC. Specify HEURISTIC for Oracle Text to check every TIMEOUT seconds if output from Outside In HTML Export has increased. The operation terminates for the document if output has not increased. An error is recorded in the CTX_USER_INDEX_ERRORS view and Oracle Text moves to the next document row to be indexed. Specify FIXED to terminate the Outside In HTML Export processing after TIMEOUT seconds regardless of whether filtering was progressing normally or just hanging. This value is useful when indexing throughput is more important than taking the time to successfully filter large documents.</td>
</tr>
</tbody>
</table>
Filter Types

Indexing Formatted Documents
To index a text column containing formatted documents such as Microsoft Word, use the INSO_FILTER. This filter automatically detects the document format. You can use the CTXSYS.INSO_FILTER system-defined preference in the parameter clause as follows:

```
create index hdocsx on hdocs(text) indextype is ctxsys.context
    parameters ('datastore ctxsys.file_datastore
                     filter ctxsys.inso_filter');
```

Explicitly Bypassing Plain Text or HTML in Mixed Format Columns
A mixed-format column is a text column containing more than one document format, such as a column that contains Microsoft Word, PDF, plain text, and HTML documents.

The INSO_FILTER can index mixed-format columns, automatically bypassing plain text, HTML, and XML documents. However, if you prefer not to depend on the built-in bypass mechanism, you can explicitly tag your rows as text and cause the INSO_FILTER to ignore the row and not process the document in any way.

The format column in the base table enables you to specify the type of document contained in the text column. 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:

```sql
create table hdocs ( 
    id number primary key, 
    fmt varchar2(10), 
    text varchar2(80) 
);
```

Assuming you are indexing mostly Word documents, you specify BINARY in the format column to filter the Word documents. Alternatively, to have the INSO_FILTER ignore an HTML document, specify TEXT in the format column.

For example, the following statements add two documents to the text table, assigning one format as BINARY and the other TEXT:

```sql
insert into hdocs values(1, 'binary', '/docs/myword.doc'); 
insert into hdocs values (2, 'text', '/docs/index.html'); 
commit;
```

To create the index, use CREATE INDEX and specify the format column name in the parameter string:

```sql
create index hdocsx on hdocs(text) indextype is ctxsys.context 
    parameters ('datastore ctxsys.file_datastore 
        filter ctxsys.inso_filter 
        format column fmt');
```

If you do not specify TEXT or BINARY for the format column, BINARY is used.

---

**Note:** You need not specify the format column in CREATE INDEX when using the INSO_FILTER.

---

**Character Set Conversion With Inso**

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

If the charset column value is not an Oracle Text character set name, the document is passed through without any character set conversion.
If you do specify the charset column and do not specify the format column, the INSO_FILTER works like the CHARSET_FILTER, except that in this case there is no Japanese character set auto-detection.

See Also: "CHARSET_FILTER" on page 2-21.

NULL_FILTER

Use the NULL_FILTER type when plain text or HTML is to be indexed and no filtering needs to be performed. NULL_FILTER has no attributes.

Indexing HTML Documents

If your document set is entirely HTML, Oracle recommends that you use the NULL_FILTER in your filter preference.

For example, to index an HTML document set, you can specify the system-defined preferences for NULL_FILTER and HTML_SECTION_GROUP as follows:

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

See Also: For more information on section groups and indexing HTML documents, see "Section Group Types" on page 2-81.

MAIL_FILTER

Use the MAIL_FILTER to transform RFC-822, RFC-2045 messages in to indexable text. The following limitations hold for the input:

- Document must be US-ASCII
- Lines must not be longer than 1024 bytes
- Document must be syntactically valid with regard to RFC-822.

Behavior for invalid input is not defined. Some deviations may be robustly handled by the filter without error. Others may result in a fetch-time or filter-time error.

The MAIL_FILTER has the following attributes:
Filter Behavior

This filter does the following for each document:

- Read and remove header fields
- Decode message body if needed, depending on Content-transfer-encoding field
- Take action depending on the Content-Type field value and the user-specified behavior in the mail filter configuration file. The possible actions are:
  - produce the body in the output text (INCLUDE)
  - INSO filter the body contents (INSOFILTER).
  - remove the body contents from the output text (IGNORE)
- If no behavior is specified for the type in the configuration file, the defaults are as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX_FIELDS</td>
<td>Specify a colon-separated list of fields to preserve in the output. These fields are transformed to tag markup. For example: From: Scott Tiger becomes: &lt;FROM&gt;Scott Tiger&lt;/FROM&gt; Only top-level files are transformed in this way.</td>
</tr>
<tr>
<td>INSO_TIMEOUT</td>
<td>Specify a timeout values for the INSO filtering invoked by the mail filter. Default is 60.</td>
</tr>
<tr>
<td>INSO_OUTPUT_FORMATTING</td>
<td>Specify either TRUE or FALSE. Default is TRUE. Specify FALSE for fast filtering of binary formatted documents. Specifying FALSE may significantly improve filtering performance; however, only minimal formatting will be preserved in the HTML output of the filter. The output will contain the necessary HTML character entities for most browsers to display it correctly. Users should evaluate the quality of the filter output when using this feature in order to determine its suitability. Note that since the output of the filter will be different compared to when this feature is not used, indexing and search results may be affected. Specify TRUE for the filter to preserve substantial amount of formatting in its HTML output when filtering binary formatted documents.</td>
</tr>
</tbody>
</table>
Filter Types

- text/*: produce body in the output text
- application/*: INSO filter the body contents
- image/*, audio/*, video/*, model/*: ignore

Multipart messages are parsed, and the mail filter applied recursively to each part. Each part is appended to the output.

All text produced will be charset-converted to the database character set, if needed.

About the Mail Filter Configuration File

The mail filter configuration file is an editable text file. Here you can override default behavior for each Content-Type. The configuration file also contains IANA to Oracle Globalization Support character set name mappings.

The location of the file must be in `ORACLE_HOME/ctx/config`. The name of the file to use is stored in the new system parameter `MAIL_FILTER_CONFIG_FILE`. On install, this is set to `drmailfl.txt`, which has useful default contents.

Oracle recommends that you create your own mail filter configuration files to avoid overwrite by the installation of a new version or patch set. The mail filter configuration file should be in the database character set.

Mail File Configuration File Structure

The file has two sections, BEHAVIOR and CHARSETS. You indicate the start of the behavior section as follows:

```
[behavior]
```

Each line following starts with a mime type, then whitespace, then behavior specification. The MIME type can be a full TYPE/SUBTYPE or just TYPE, which will apply to all subtypes of that type. TYPE/SUBTYPE specification overrides TYPE specification, which overrides default behavior. Behavior can be INCLUDE, INSOFILTER, or IGNORE (see "Filter Behavior" on page 2-28 for definitions). For instance:

```
application/zip   IGNORE
application/msword INSOFILTER
model             IGNORE
```

You cannot specify behavior for "multipart" or "message" types. If you do, such lines are ignored. Duplicate specification for a type replaces earlier specifications.
Comments can be included in the mail configuration file by starting lines with the # symbol.

The charset mapping section begins with

[charsets]

Lines consist of an IANA name, then whitespace, then a Oracle Globalization Support charset name, like:

US-ASCII US7ASCII
ISO-8859-1 WE8ISO8859P1

This file is the only way the mail filter gets the mappings. There are no defaults. When you change the configuration file, the changes affect only the documents indexed after that point. You must flush the shared pool after changing the file.

USER_FILTER

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>Specify the name of the filter executable.</td>
</tr>
</tbody>
</table>

**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.

On UNIX, the executable you specify must exist in the $ORACLE_HOME/ctx/bin directory. On Windows, the executable you specify must exist in the %ORACLE_HOME%/bin directory.

You must create your user-filter executable with two parameters: the first is the name of the input file to be read, and the second is the name of the output file to be written to.

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

The following example Perl script to be used as the user filter. This script converts the input text file specified in the first argument to uppercase and writes the output to the location specified in the second argument:

```perl
#!/usr/local/bin/perl

open(IN, $ARGV[0]);
open(OUT, ">".$ARGV[1]);

while (<IN>)
{
  tr/a-z/A-Z/;
  print OUT;
}

close (IN);
close (OUT);
```

Assuming that this file is named `upcase.pl`, create the filter preference as follows:

```
begin
  ctx_ddl.create_preference
  {
    preference_name => 'USER_FILTER_PREF',
    object_name => 'USER_FILTER'
  };
  ctx_ddl.set_attribute
  ("USER_FILTER_PREF","COMMAND","upcase.pl");
end;
```

Create the index in SQL*Plus as follows:

```
create index user_filter_idx on user_filter ( docs )
  indextype is ctxsys.context
  parameters ('FILTER USER_FILTER_PREF');
```

PROCEDURE_FILTER

Use the `PROCEDURE_FILTER` type to filter your documents with a stored procedure. The stored procedure is called each time a document needs to be filtered. This type has the following attributes:
Filter Types

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Purpose</th>
<th>Allowable Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure</td>
<td>Name of the filter stored procedure.</td>
<td>Any procedure. The procedure can be PL/SQL stored procedure.</td>
</tr>
<tr>
<td>input_type</td>
<td>Type of input argument for stored procedure.</td>
<td>VARCHAR2, BLOB, CLOB, FILE</td>
</tr>
<tr>
<td>output_type</td>
<td>Type of output argument for stored procedure.</td>
<td>VARCHAR2, CLOB, FILE</td>
</tr>
<tr>
<td>rowid_parameter</td>
<td>Include rowid parameter?</td>
<td>TRUE/FALSE</td>
</tr>
<tr>
<td>format_parameter</td>
<td>Include format parameter?</td>
<td>TRUE/FALSE</td>
</tr>
<tr>
<td>charset_parameter</td>
<td>Include charset parameter?</td>
<td>TRUE/FALSE</td>
</tr>
</tbody>
</table>

**procedure**

Specify the name of the stored procedure to use for filtering. The procedure can be a PL/SQL stored procedure. The procedure can be a safe callout or call a safe callout.

With the rowid_parameter, format_parameter, and charset_parameter set to FALSE, the procedure can have one of the following signatures:

PROCEDURE(IN BLOB, IN OUT NOCOPY CLOB)
PROCEDURE(IN CLOB, IN OUT NOCOPY CLOB)
PROCEDURE(IN VARCHAR, IN OUT NOCOPY CLOB)
PROCEDURE(IN BLOB, IN OUT NOCOPY VARCHAR2)
PROCEDURE(IN CLOB, IN OUT NOCOPY VARCHAR2)
PROCEDURE(IN VARCHAR2, IN OUT NOCOPY VARCHAR2)
PROCEDURE(IN BLOB, IN VARCHAR2)
PROCEDURE(IN CLOB, IN VARCHAR2)
PROCEDURE(IN VARCHAR2, IN VARCHAR2)

The first argument is the content of the unfiltered row as passed out by the datastore. The second argument is for the procedure to pass back the filtered document text.

The procedure attribute is mandatory and has no default.

**input_type**

Specify the type of the input argument of the filter procedure. You can specify one of the following:
### Filter Types

The **input_type** attribute is not mandatory. If not specified, BLOB is the default.

**output_type**  
Specify the type of output argument of the filter procedure. You can specify one of the following types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOB</td>
<td>The input argument is of type CLOB. The unfiltered document is contained in the CLOB passed in. No pre-filtering or character set conversion is done. If the datastore outputs binary data, that binary data is written directly to the CLOB, with Globalization Support doing implicit mapping to character data as best it can.</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>The input argument is of type VARCHAR2. The unfiltered document is contained in the VARCHAR2 passed in. The document can be a maximum of 32767 bytes of data. If the unfiltered document is greater than this length, an error is raised for the document and the filter procedure is not called.</td>
</tr>
<tr>
<td>FILE</td>
<td>The input argument is of type VARCHAR2. The unfiltered document content is contained in a temporary file in the file system whose filename is stored in the VARCHAR2 passed in. For example, the value of the passed-in VARCHAR2 might be <code>tmp/mydoc.tmp</code> which means that the document content is stored in the file <code>/tmp/mydoc.tmp</code>. The file input type is useful only when your procedure is a safe callout, which can read the file.</td>
</tr>
<tr>
<td>BLOB</td>
<td>The output argument is IN OUT NOCOPY CLOB. Your procedure must write the filtered content to the CLOB passed in.</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>The output argument is IN OUT NOCOPY VARCHAR2. Your procedure must write the filtered content to the VARCHAR2 variable passed in.</td>
</tr>
</tbody>
</table>
The output_type attribute is not mandatory. If not specified, CLOB is the default.

**rowid parameter**
When you specify TRUE, the rowid of the document to be filtered is passed as the first parameter, before the input and output parameters.

For example, with INPUT_TYPE BLOB, OUTPUT_TYPE CLOB, and ROWID_PARAMETER TRUE, the filter procedure must have the signature as follows:

```
procedure(in rowid, in blob, in out nocopy clob)
```

This attribute is useful for when your procedure requires data from other columns or tables. This attribute is not mandatory. The default is FALSE.

**format parameter**
When you specify TRUE, the value of the format column of the document being filtered is passed to the filter procedure before input and output parameters, but after the rowid parameter, if enabled.

You specify the name of the format column at index time in the parameters string, using the keyword 'format column <columnname>'. The parameter type must be IN VARCHAR2.

The format column value can be read by means of the rowid parameter, but this attribute enables a single filter to work on multiple table structures, because the format attribute is abstracted and does not require the knowledge of the name of the table or format column.

**FORMAT_PARAMETER** is not mandatory. The default is FALSE.

**charset parameter**
When you specify TRUE, the value of the charset column of the document being filtered is passed to the filter procedure before input and output parameters, but after the rowid and format parameter, if enabled.

---

**Type** | **Description**
---|---
FILE | The output argument must be IN VARCHAR2. On entering the filter procedure, the output argument is the name of a temporary file. The filter procedure must write the filtered contents to this named file.
| Using a FILE output type is useful only when the procedure is a safe callout, which can write to the file.

---

The output_type attribute is not mandatory. If not specified, CLOB is the default.
You specify the name of the charset column at index time in the parameters string, using the keyword 'charset column <columnname>'. The parameter type must be IN VARCHAR2.

CHARSET_PARAMETER attribute is not mandatory. The default is FALSE.

**Parameter Order**

ROWID_PARAMETER, FORMAT_PARAMETER, and CHARSET_PARAMETER are all independent. The order is rowid, the format, then charset, but the filter procedure is passed only the minimum parameters required.

For example, assume that INPUT_TYPE is BLOB and OUTPUT_TYPE is CLOB. If your filter procedure requires all parameters, the procedure signature must be:

```
(id IN ROWID, format IN VARCHAR2, charset IN VARCHAR2, input IN BLOB, output IN OUT NOCOPY CLOB)
```

If your procedure requires only the ROWID, then the procedure signature must be:

```
(id IN ROWID, input IN BLOB, output IN OUT NOCOPY CLOB)
```

**Procedure Filter Execute Requirements**

In order to create an index using a PROCEDURE_FILTER preference, the index owner must have execute permission on the procedure.

**Error Handling**

The filter procedure can raise any errors needed through the normal PL/SQL raise_application_error facility. These errors are propagated to the CTX_USER_INDEX_ERRORS view or reported to the user, depending on how the filter is invoked.

**Procedure Filter Preference Example**

Consider a filter procedure CTXSYS.NORMALIZE that you define with the following signature:

```
PROCEDURE NORMALIZE(id IN ROWID, charset IN VARCHAR2, input IN CLOB, output IN OUT NOCOPY VARCHAR2);
```

To use this procedure as your filter, set up your filter preference as follows:

```
begin
    ctx_ddl.create_preference('myfilt', 'procedure_filter');
    ctx_ddl.set_attribute('myfilt', 'procedure', 'normalize');
end;
```
ctx_ddl.set_attribute('myfilt', 'input_type', 'clob');
ctx_ddl.set_attribute('myfilt', 'output_type', 'varchar2');
ctx_ddl.set_attribute('myfilt', 'rowid_parameter', 'TRUE');
ctx_ddl.set_attribute('myfilt', 'charset_parameter', 'TRUE');
end;

Lexer Types

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

<table>
<thead>
<tr>
<th>type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASIC_LEXER</td>
<td>Lexer for extracting tokens from text in languages, such as English and most western European languages that use white space delimited words.</td>
</tr>
<tr>
<td>MULTI_LEXER</td>
<td>Lexer for indexing tables containing documents of different languages</td>
</tr>
<tr>
<td>CHINESE_VGRAM_LEXER</td>
<td>Lexer for extracting tokens from Chinese text.</td>
</tr>
<tr>
<td>CHINESE_LEXER</td>
<td>Lexer for extracting tokens from Chinese text.</td>
</tr>
<tr>
<td>JAPANESE_VGRAM_LEXER</td>
<td>Lexer for extracting tokens from Japanese text.</td>
</tr>
<tr>
<td>JAPANESE_LEXER</td>
<td>Lexer for extracting tokens from Japanese text.</td>
</tr>
<tr>
<td>KOREAN_LEXER</td>
<td>Lexer for extracting tokens from Korean text.</td>
</tr>
<tr>
<td>KOREAN_MORPH_LEXER</td>
<td>Lexer for extracting tokens from Korean text (recommended).</td>
</tr>
<tr>
<td>USER_LEXER</td>
<td>Lexer you create to index a particular language.</td>
</tr>
<tr>
<td>WORLD_LEXER</td>
<td>Lexer for indexing tables containing documents of different languages; autodetects languages in a document</td>
</tr>
</tbody>
</table>

BASIC_LEXER

Use the BASIC_LEXER type to identify tokens for creating Text indexes for English and all other supported whitespace delimited languages.

The BASIC_LEXER also enables base-letter conversion, composite word indexing, case-sensitive indexing and alternate spelling for whitespace delimited languages that have extended character sets.

In English and French, you can use the BASIC_LEXER to enable theme indexing.
**Note:** Any processing the lexer does to tokens before indexing (for example, removal of characters, and base-letter conversion) are also performed on query terms at query time. This ensures that the query terms match the form of the tokens in the Text index.

BASICLexer supports any database character set.

BASICLexer has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>continuation</td>
<td>characters</td>
</tr>
<tr>
<td>numgroup</td>
<td>characters</td>
</tr>
<tr>
<td>numjoin</td>
<td>characters</td>
</tr>
<tr>
<td>printjoints</td>
<td>characters</td>
</tr>
<tr>
<td>punctuations</td>
<td>characters</td>
</tr>
<tr>
<td>skipjoints</td>
<td>characters</td>
</tr>
<tr>
<td>startjoints</td>
<td>non alphanumeric characters that occur at the beginning of a token (string)</td>
</tr>
<tr>
<td>endjoints</td>
<td>non alphanumeric characters that occur at the end of a token (string)</td>
</tr>
<tr>
<td>whitespace</td>
<td>characters</td>
</tr>
<tr>
<td>newline</td>
<td>NEWLINE (\n)</td>
</tr>
<tr>
<td></td>
<td>CARRIAGE_RETURN (\r)</td>
</tr>
<tr>
<td>base_letter</td>
<td>NO (disabled)</td>
</tr>
<tr>
<td></td>
<td>YES (enabled)</td>
</tr>
<tr>
<td>base_letter_type</td>
<td>GENERIC (default)</td>
</tr>
<tr>
<td></td>
<td>SPECIFIC</td>
</tr>
<tr>
<td>override_base_letter</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE (default)</td>
</tr>
<tr>
<td>mixed_case</td>
<td>NO (disabled)</td>
</tr>
<tr>
<td></td>
<td>YES (enabled)</td>
</tr>
<tr>
<td>composite</td>
<td>DEFAULT (no composite word indexing, default)</td>
</tr>
</tbody>
</table>
continuation
Specify the characters that indicate a word continues on the next line and should be indexed as a single token. The most common continuation characters are hyphen ‘-’ and backslash ‘\’.
**numgroup**
Specify a single character that, when it appears in a string of digits, indicates that the digits are groupings within a larger single unit.

For example, comma ',' might be defined as a numgroup character because it often indicates a grouping of thousands when it appears in a string of digits.

**numjoin**
Specify the characters that, when they appear in a string of digits, cause Oracle Text to index the string of digits as a single unit or word.

For example, period '.' can be defined as numjoin characters because it often serves as decimal points when it appears in a string of digits.

---

**Note:** The default values for numjoin and numgroup are determined by the Globalization Support initialization parameters that are specified for the database.

In general, a value need not be specified for either numjoin or numgroup when creating a lexer preference for **BASIC_LEXER**.

---

**printjoins**
Specify the non alphanumeric characters that, when they appear anywhere in a word (beginning, middle, or end), are processed as alphanumeric and included with the token in the Text index. This includes printjoins that occur consecutively.

For example, if the hyphen '-' and underscore '_' characters are defined as printjoins, terms such as *pseudo-intellectual* and *_file_* are stored in the Text index as *pseudo-intellectual* and *_file_*.

---

**Note:** If a printjoins character is also defined as a punctuations character, the character is only processed as an alphanumeric character if the character immediately following it is a standard alphanumeric character or has been defined as a printjoins or skipjoins character.

---

**punctuations**
Specify the non-alphanumeric characters that, when they appear at the end of a word, indicate the end of a sentence. The defaults are period '.', question mark '?', and exclamation point '!'.

---
Characters that are defined as punctuations are removed from a token before text indexing. However, if a punctuations character is also defined as a printjoins character, the character is removed only when it is the last character in the token.

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:

<table>
<thead>
<tr>
<th>Token</th>
<th>Indexed Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>.doc</td>
<td>.doc</td>
</tr>
<tr>
<td>dog.doc</td>
<td>dog.doc</td>
</tr>
<tr>
<td>dog..doc</td>
<td>dog..doc</td>
</tr>
<tr>
<td>dog.</td>
<td>dog</td>
</tr>
<tr>
<td>dog...</td>
<td>dog..</td>
</tr>
</tbody>
</table>

In addition, BASICLexer uses punctuations characters in conjunction with newline and whitespace characters to determine sentence and paragraph delimiters for sentence/paragraph searching.

**skipjoins**
Specify the non-alphanumeric characters that, when they appear within a word, identify the word as a single token; however, the characters are not stored with the token in the Text index.

For example, if the hyphen character '-' is defined as a skipjoins, the word *pseudo-intellectual* is stored in the Text index as *pseudointellectual*.

**Note:** printjoins and skipjoins are mutually exclusive. The same characters cannot be specified for both attributes.

**startjoins/endjoins**
For startjoins, specify the characters that when encountered as the first character in a token explicitly identify the start of the token. The character, as well as any other startjoins characters that immediately follow it, is included in the Text index entry for the token. In addition, the first startjoins character in a string of startjoins characters implicitly ends the previous token.

For endjoins, specify the characters that when encountered as the last character in a token explicitly identify the end of the token. The character, as well as any other
**Lexer Types**

*startjoins* characters that immediately follow it, is included in the Text index entry for the token.

The following rules apply to both *startjoins* and *endjoins*:

- The characters specified for *startjoins*/*endjoins* cannot occur in any of the other attributes for `BASIC_LEXER`.
- *startjoins*/*endjoins* characters can occur only at the beginning or end of tokens

**whitespace**

Specify the characters that are treated as blank spaces between tokens. `BASIC_LEXER` uses whitespace characters in conjunction with punctuations and newline characters to identify character strings that serve as sentence delimiters for sentence and paragraph searching.

The predefined default values for *whitespace* are 'space' and 'tab'. These values cannot be changed. Specifying characters as *whitespace* characters adds to these defaults.

**newline**

Specify the characters that indicate the end of a line of text. `BASIC_LEXER` uses newline characters in conjunction with punctuations and whitespace characters to identify character strings that serve as paragraph delimiters for sentence and paragraph searching.

The only valid values for *newline* are `NEWLINE` and `CARRIAGE_RETURN` (for carriage returns). The default is `NEWLINE`.

**base_letter**

Specify whether characters that have diacritical marks (umlauts, cedillas, acute accents, and so on) are converted to their base form before being stored in the Text index. The default is NO (base-letter conversion disabled). For more information on base-letter conversions and *base_letter_type*, see Base-Letter Conversion on page 15-2.

**base_letter_type**

Specify GENERIC or SPECIFIC.

The GENERIC value is the default and means that base letter transformation uses one transformation table that applies to all languages. For more information on base-letter conversions and *base_letter_type*, see Base-Letter Conversion on page 15-2.
override_base_letter
When base_letter is enabled at the same time as alternate_spelling, it is sometimes necessary to override base_letter to prevent unexpected results from serial transformations. See Overriding Base-Letter Transformations with Alternate Spelling on page 15-4. Default is FALSE.

mixed_case
Specify whether the lexer leaves the tokens exactly as they appear in the text or converts the tokens to all uppercase. The default is NO (tokens are converted to all uppercase).

Note: Oracle Text ensures that word queries match the case sensitivity of the index being queried. As a result, if you enable case sensitivity for your Text index, queries against the index are always case sensitive.

composite
Specify whether composite word indexing is disabled or enabled for either GERMAN or DUTCH text. The default is DEFAULT (composite word indexing disabled).

Words that are usually one entry in a German dictionary are not split into composite stems, while words that aren’t dictionary entries are split into composite stems.

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

Stemming User-Dictionaries
Oracle Text ships with a system stemming dictionary ($ORACLE_HOME/ctx/data/enlx/dren.dct), which is used for both ENGLISH and DERIVATIONAL stemming. You can create a user-dictionary for your own language to customize how words are decomposed. These dictionaries are shown in Table 2–1.
Stemming user-dictionaries are not supported for languages other than those listed in Table 2–1.

The format for the user dictionary is as follows:

input term <tab> output term

The individual parts of the decomposed word must be separated by the # character. The following example entries are for the German word Hauptbahnhof:

Hauptbahnhof<tab>Haupt#Bahnhof
Hauptbahnhofes<tab>Haupt#Bahnhof
Hauptbahnhof<tab>Haupt#Bahnhof
Hauptbahnhoef<tab>Haupt#Bahnhof

**index_themes**

Specify YES to index theme information in English or French. This makes ABOUT queries more precise. The index_themes and index_text attributes cannot both be NO.

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

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

**prove_themes**

Specify YES to prove themes. Theme proving attempts to find related themes in a document. When no related themes are found, parent themes are eliminated from the document.

While theme proving is acceptable for large documents, short text descriptions with a few words rarely prove parent themes, resulting in poor recall performance with ABOUT queries.

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ORACLE_HOME/ctx/data/frlx/dfrfr.dct</td>
<td>French</td>
</tr>
<tr>
<td>$ORACLE_HOME/ctx/data/delx/drde.dct</td>
<td>German</td>
</tr>
<tr>
<td>$ORACLE_HOME/ctx/data/nllx/drn1.dct</td>
<td>Dutch</td>
</tr>
<tr>
<td>$ORACLE_HOME/ctx/data/itlx/drit.dct</td>
<td>Italian</td>
</tr>
<tr>
<td>$ORACLE_HOME/ctx/data/eslx/dres.dct</td>
<td>Spanish</td>
</tr>
</tbody>
</table>
Theme proving results in higher precision and less recall (less rows returned) for ABOUT queries. For higher recall in ABOUT queries and possibly less precision, you can disable theme proving. Default is YES.

The prove_themes attribute is supported for CONTEXT and CTXRULE indexes.

**theme_language**
Specify which knowledge base to use for theme generation when index_themes is set to YES. When index_themes is NO, setting this parameter has no effect on anything.

You can specify any Globalization Support language or AUTO. You must have a knowledge base for the language you specify. This release provides a knowledge base in only English and French. In other languages, you can create your own knowledge base.

**See Also:** "Adding a Language-Specific Knowledge Base" in Chapter 14, "Executables".

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

**index_stems**
Specify the stemmer to use for stem indexing. You can choose one of

- NONE
- ENGLISH
- DERIVATIONAL
- DUTCH
- FRENCH
- GERMAN
- SPANISH

Tokens are stemmed to a single base form at index time in addition to the normal forms. Indexing stems enables better query performance for stem ($) queries, such as $computed.

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

The default is NO.
alternate_spelling
Specify either GERMAN, DANISH, or SWEDISH to enable the alternate spelling in one of these languages. Enabling alternate spelling enables you to query a word in any of its alternate forms.

Alternate spelling is off by default; however, in the language-specific scripts that Oracle provides in admin/defaults (drdefd.sql for German, drdefdk.sql for Danish, and drdefs.sql for Swedish), alternate spelling is turned on. If your installation uses these scripts, then alternate spelling is on. However, you can specify NONE for no alternate spelling. For more information about the alternate spelling conventions Oracle Text uses, see Alternate Spelling on page 15-2.

new_german_spelling
Specify whether the queries using the BASIC_LEXER return both traditional and reformed (new) spellings of German words. If new_german_spelling is set to YES, then both traditional and new forms of words are indexed. If it is set to NO, then the word will be indexed only as it is provided in the query. The default is NO.

See Also:  “New German Spelling”  on page 15-3

BASIC_LEXER Example
The following example sets printjoin characters and disables theme indexing with the BASIC_LEXER:

begin
ctx_ddl.create_preference('mylex', 'BASIC_LEXER');
ctx_ddl.set_attribute('mylex', 'printjoins', '_-');
ctx_ddl.set_attribute ( 'mylex', 'index_themes', 'NO' );
ctx_ddl.set_attribute ( 'mylex', 'index_text', 'YES' );
end;

To create the index with no theme indexing and with printjoins characters set as described, issue the following statement:

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

Use MULTI_LEXER to index text columns that contain documents of different languages. For example, you can use this lexer to index a text column that stores English, German, and Japanese documents.

This lexer has no attributes.

You must have a language column in your base table. To index multi-language tables, you specify the language column when you create the index.

You create a multi-lexer preference with the CTX_DDL.CREATE_PREFERENCE. You add language-specific lexers to the multi-lexer preference with the CTX_DDL.ADD_SUB_LEXER procedure.

During indexing, the MULTI_LEXER examines each row's language column value and switches in the language-specific lexer to process the document.

The WORLD_LEXER lexer also performs multi-language indexing, but without the need for separate language columns (that is, it has automatic language detection). For more on WORLD_LEXER, see "WORLD_LEXER" on page 2-71.

**Multi-language Stoplists**

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

See Also: "Multi-Language Stoplists" on page 2-89.

**MULTI_LEXER Example**

Create the multi-language table with a primary key, a text column, and a language column as follows:

```sql
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:

```sql
ctx_ddl.create_preference('english_lexer','basic_lexer');
ctx_ddl.set_attribute('english_lexer','index_themes','yes');
ctx_ddl.set_attribute('english_lexer','theme_language','english');
```
ctx_ddl.create_preference('german_lexer','basic_lexer');
ctx_ddl.set_attribute('german_lexer','composite','german');
ctx_ddl.set_attribute('german_lexer','mixed_case','yes');
ctx_ddl.set_attribute('german_lexer','alternate_spelling','german');

ctx_ddl.create_preference('japanese_lexer','japanese_vgram_lexer');

Create the multi-lexer preference:
ctx_ddl.create_preference('global_lexer', 'multi_lexer');

Since the stored documents are mostly English, make the English lexer the default using CTX_DDL.ADD_SUB_LEXER:
ctx_ddl.add_sub_lexer('global_lexer','default','english_lexer');

Now add the German and Japanese lexers in their respective languages with CTX_DDL.ADD_SUB_LEXER procedure. Also assume that the language column is expressed in the standard ISO 639-2 language codes, so add those as alternate values.
ctx_ddl.add_sub_lexer('global_lexer','german','german_lexer','ger');
ctx_ddl.add_sub_lexer('global_lexer','japanese','japanese_lexer','jpn');

Now create the index globalx, specifying the multi-lexer preference and the language column in the parameter clause as follows:
create index globalx on globaldoc(text) indextype is ctxsys.context parameters ('lexer global_lexer language column lang');

Querying Multi-Language Tables
At query time, the multi-lexer examines the language setting and uses the sub-lexer preference for that language to parse the query. If the language is not set, then the default lexer is used.
Otherwise, the query is parsed and run as usual. The index contains tokens from multiple languages, so such a query can return documents in several languages. To limit your query to a given language, use a structured clause on the language column.
**CHINESE_VGRAM_LEXER**

The `CHINESE_VGRAM_LEXER` type identifies tokens in Chinese text for creating Text indexes. It has no attributes.

**Character Sets**
You can use this lexer if your database character set is one of the following:
- AL32UTF8
- ZHS16CGB231280
- ZHS16GBK
- ZHS32GB18030
- ZHT32EUC
- ZHT16BIG5
- ZHT32TRIS
- ZHT16MSWIN950
- ZHT16HKSCS
- UTF8

**CHINESE_LEXER**

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

This lexer offers the following benefits over the `CHINESE_VGRAM_LEXER`:
- generates a smaller index
- better query response time
- generates real word tokens resulting in better query precision
- supports stop words

Because the `CHINESE_LEXER` uses a different algorithm to generate tokens, indexing time is longer than with `CHINESE_VGRAM_LEXER`.

You can use this lexer if your database character is one of the Chinese or Unicode character sets supported by Oracle.
Customizing the Chinese Lexicon
You can modify the existing lexicon (dictionary) used by the Chinese lexer, or create your own Chinese lexicon, with the ctxlc command.

See Also: Lexical Compiler (ctxlc) in Executables

JAPANESE_VGRAM_LEXER
The JAPANESE_VGRAM_LEXER type identifies tokens in Japanese for creating Text indexes. It has no attributes. This lexer supports the stem ($) operator.

JAPANESE_VGRAM_LEXER Attribute
This lexer has the following attribute:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>delimiter</td>
<td>Specify NONE or ALL to ignore certain Japanese blank characters, such as a full-width forward slash or a full-width middle dot. Default is NONE.</td>
</tr>
</tbody>
</table>

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

- JA16SJIS
- JA16EUC
- UTF8
- AL32UTF8
- JA16EUCTILDE
- JA16EUCYEN
- JA16SJISTILDE
- JA16SJISYEN

JAPANESE_LEXER
The JAPANESE_LEXER type identifies tokens in Japanese for creating Text indexes. This lexer supports the stem ($) operator.

This lexer offers the following benefits over the JAPANESE_VGRAM_LEXER:
generates a smaller index
- better query response time
- generates real word tokens resulting in better query precision

Because the JAPANESE_LEXER uses a new algorithm to generate tokens, indexing time is longer than with JAPANESE_VGRAM_LEXER.

Customizing the Japanese Lexicon
You can modify the existing lexicon (dictionary) used by the Japanese lexer, or create your own Japanese lexicon, with the ctxlc command.

See Also: Lexical Compiler (ctxlc) in Executables

JAPANESE_LEXER Attribute
This lexer has the following attribute:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>delimiter</td>
<td>Specify NONE or ALL to ignore certain Japanese blank characters, such as a full-width forward slash or a full-width middle dot. Default is NONE.</td>
</tr>
</tbody>
</table>

JAPANESE LEXER Character Sets
The JAPANESE_LEXER supports the following character sets:

- JA16SJIS
- JA16EUC
- UTF8
- AL32UTF8
- JA16EUCTILDE
- JA16EUCYEN
- JA16SJISTILDE
- JA16SJISYEN

Japanese Lexer Example
When you specify JAPANESE_LEXER for creating text index, the JAPANESE_LEXER resolves a sentence into words.
For example, the following compound word (*natural language institute*)

'自然言語処理'

is indexed as three tokens:

'自然', '言語', '処理'

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

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

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

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

- KO16KSC5601
- UTF8

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>verb</td>
<td>Specify TRUE or FALSE to index verbs. Default is TRUE.</td>
</tr>
</tbody>
</table>
**KOREAN_MORPH_LEXER**

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

- better morphological analysis of Korean text
- faster indexing
- smaller indexes
- more accurate query searching
- support for AL32UTF8 character set

**Supplied Dictionaries**
The **KOREAN_MORPH_LEXER** uses four dictionaries:

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>$ORACLE_HOME/ctx/data/kolx/drk2sdic.dat</td>
</tr>
</tbody>
</table>

**Limitations**
Sentence and paragraph sections are not supported with the Korean lexer.
The grammar, user-defined, and stopword dictionaries should be written using the KSC 5601 or MSWIN949 character sets. You can modify these dictionaries using the defined rules. The system dictionary must not be modified.

You can add unregistered words to the user-defined dictionary file. The rules for specifying new words are in the file.

Supported Character Sets

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

- KO16KSC5601
- KO16MSWIN949
- UTF8
- AL32UTF8

Unicode Support

The KOREAN_MORPH_LEXER supports:

- words in non-KSC5601 Korean characters defined in Unicode
- supplementary characters

See Also: For information on supplementary characters, see the Oracle Database Globalization Support Guide

Some Korean documents may have non-KSC5601 characters in them. As the KOREAN_MORPH_LEXER can recognize all possible 11,172 Korean (Hangul) characters, such documents can also be interpreted by using the UTF8 or AL32UTF8 character sets.

Use the AL32UTF8 character set for your database to extract surrogate characters. By default, the KOREAN_MORPH_LEXER extracts all series of surrogate characters in a document as one token for each series.
Limitations on Korean Unicode Support

For conversion Hanja to Hangul (Korean), the KOREAN_MORPH_LEXER supports only the 4888 Hanja characters defined in KSC5601.

KOREAN_MORPH_LEXER Attributes

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>verb_adjective</td>
<td>Specify TRUE or FALSE to index verbs and adjectives. Default is FALSE.</td>
</tr>
<tr>
<td>one_char_word</td>
<td>Specify TRUE or FALSE to index one syllable. Default is FALSE.</td>
</tr>
<tr>
<td>number</td>
<td>Specify TRUE or FALSE to index number. Default is FALSE.</td>
</tr>
<tr>
<td>user_dic</td>
<td>Specify TRUE or FALSE to index user dictionary. Default is TRUE.</td>
</tr>
<tr>
<td>stop_dic</td>
<td>Specify TRUE or FALSE to use stop-word dictionary. Default is TRUE. The stop-word dictionary belongs to KOREAN_MORPH_LEXER.</td>
</tr>
<tr>
<td>composite</td>
<td>Specify indexing style of composite noun. Specify COMPOSITE_ONLY to index only composite nouns. Specify NGRAM to index all noun components of a composite noun. Specify COMPONENT_WORD to index single noun components of composite nouns as well as the composite noun itself. Default is COMPONENT_WORD. The following example describes the difference between NGRAM and COMPONENT_WORD.</td>
</tr>
<tr>
<td>morpheme</td>
<td>Specify TRUE or FALSE for morphological analysis. If set to FALSE, tokens are created from the words that are divided by delimiters such as white space in the document. Default is TRUE.</td>
</tr>
<tr>
<td>to_upper</td>
<td>Specify TRUE or FALSE to convert English to uppercase. Default is TRUE.</td>
</tr>
<tr>
<td>hanja</td>
<td>Specify TRUE to index hanja characters. If set to FALSE, hanja characters are converted to hangul characters. Default is FALSE.</td>
</tr>
<tr>
<td>long_word</td>
<td>Specify TRUE to index long words that have more than 16 syllables in Korean. Default is FALSE.</td>
</tr>
<tr>
<td>japanese</td>
<td>Specify TRUE to index Japanese characters in Unicode (only in the 2-byte area). Default is FALSE.</td>
</tr>
<tr>
<td>english</td>
<td>Specify TRUE to index alphanumeric strings. Default is TRUE.</td>
</tr>
</tbody>
</table>
Limitations
Sentence and paragraph sections are not supported with the Korean lexer.

**KOREAN_MORPH_LEXER Example: Setting Composite Attribute**
You can use the composite attribute to control how composite nouns are indexed.

**NGRAM Example** When you specify NGRAM for the composite attribute, composite nouns are indexed with all possible component tokens. For example, the following composite noun (information processing institute)

‘정보처리학회’

is indexed as six tokens:


‘처리학회’, ‘정보처리학회’

You can specify NGRAM indexing as follows:

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

To create the index:

create index koreannx on korean(text) indextype is ctxsys.context
parameters ('lexer korean_lexer');

**COMPONENT_WORD Example** When you specify COMPONENT_WORD for the composite attribute, composite nouns and their components are indexed. For example, the following composite noun (information processing institute)

‘정보처리학회’

is indexed as four tokens:

‘정보처리학회’
You can specify `COMPONENT_WORD` indexing as follows:

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

To create the index:

```sql
create index koreanx on korean(text) indextype is ctxsys.context
parameters ('lexer korean_lexer');
```

**USER_LEXER**

Use `USER_LEXER` to plug in your own language specific lexing solution. This enables you to define lexers for languages that are not supported by Oracle Text. It also enables you to define a new lexer for a language that is supported but whose lexer is inappropriate for your application.

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

<table>
<thead>
<tr>
<th>User-define Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing Procedure</td>
<td>Stored procedure (PL/SQL) which implements the tokenization of documents and stop words. Output must be an XML document as specified in this section.</td>
</tr>
<tr>
<td>Query Procedure</td>
<td>Stored procedure (PL/SQL) which implements the tokenization of query words. Output must be a XML document as specified in this section.</td>
</tr>
</tbody>
</table>

**Limitations**

The following features are not supported with the `USER_LEXER`:

- `CTX_DOC.GIST` and `CTX_DOC.THEMES`
- `CTX_QUERY.HFEEDBACK`
- `ABOUT` query operator
- `CTXRULE` indextype
Lexer Types

- VGRAM indexing algorithm

**USER_LEXER Attributes**
The USER_LEXER has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Supported Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX_PROCEDURE</td>
<td>Name of a stored procedure. No default provided.</td>
</tr>
<tr>
<td>INPUT_TYPE</td>
<td>VARCHAR2, CLOB. Default is CLOB.</td>
</tr>
<tr>
<td>QUERY_PROCEDURE</td>
<td>Name of a stored procedure. No default provided.</td>
</tr>
</tbody>
</table>

**INDEX_PROCEDURE**
This callback stored procedure is called by Oracle Text as needed to tokenize a document or a stop word found in the stoplist object.

**Requirements**
This procedure can be a PL/SQL stored procedure. The index owner must have EXECUTE privilege on this stored procedure. This stored procedure must not be replaced or dropped after the index is created. You can replace or drop this stored procedure after the index is dropped.

**Parameters**
Two different interfaces are supported for the user-defined lexer indexing procedure:
- VARCHAR2 Interface
- CLOB Interface

**Restrictions**
This procedure must not perform any of the following operations:
- rollback
- explicitly or implicitly commit the current transaction
- issue any other transaction control statement
- alter the session language or territory

The child elements of the root element tokens of the XML document returned must be in the same order as the tokens occur in the document or stop word being tokenized.
The behavior of this stored procedure must be deterministic with respect to all parameters.

**INPUT_TYPE**

Two different interfaces are supported for the User-defined lexer indexing procedure. One interface enables the document or stop word and the corresponding tokens encoded as XML to be passed as VARCHAR2 datatype whereas the other interface uses the CLOB datatype. This attribute indicates the interface implemented by the stored procedure specified by the INDEX_PROCEDURE attribute.

**VARCHAR2 Interface** BASIC_WORDLIST Attributes Table 2-2 describes the interface that enables the document or stop word from stoplist object to be tokenized to be passed as VARCHAR2 from Oracle Text to the stored procedure and for the tokens to be passed as VARCHAR2 as well from the stored procedure back to Oracle Text.

Your user-defined lexer indexing procedure should use this interface when all documents in the column to be indexed are smaller than or equal to 32512 bytes and the tokens can be represented by less than or equal to 32512 bytes. In this case the CLOB interface given in Table 2–3 can also be used, although the VARCHAR2 interface will generally perform faster than the CLOB interface.

This procedure must be defined with the following parameters:
Table 2–2  VARCHAR2 Interface for INDEX_PROCEDURES

<table>
<thead>
<tr>
<th>Parameter Position</th>
<th>Parameter Mode</th>
<th>Parameter Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IN</td>
<td>VARCHAR2</td>
<td>Document or stop word from stoplist object to be tokenized. If the document is larger than 32512 bytes then Oracle Text will report a document level indexing error.</td>
</tr>
<tr>
<td>2</td>
<td>IN OUT</td>
<td>VARCHAR2</td>
<td>Tokens encoded as XML. If the document contains no tokens, then either NULL must be returned or the tokens element in the XML document returned must contain no child elements. Byte length of the data must be less than or equal to 32512. To improve performance, use the NOCOPY hint when declaring this parameter. This passes the data by reference, rather than passing data by value. The XML document returned by this procedure should not include unnecessary whitespace characters (typically used to improve readability). This reduces the size of the XML document which in turn minimizes the transfer time. To improve performance, index_procedure should not validate the XML document with the corresponding XML schema at run-time. Note that this parameter is IN OUT for performance purposes. The stored procedure has no need to use the IN value.</td>
</tr>
<tr>
<td>3</td>
<td>IN</td>
<td>BOOLEAN</td>
<td>Oracle Text sets this parameter to TRUE when Text needs the character offset and character length of the tokens as found in the document being tokenized. Oracle Text sets this parameter to FALSE when Text is not interested in the character offset and character length of the tokens as found in the document being tokenized. This implies that the XML attributes off and len must not be used.</td>
</tr>
</tbody>
</table>

CLOB Interface  Table 2–3 describes the CLOB interface that enables the document or stop word from stoplist object to be tokenized to be passed as CLOB from Oracle Text to the stored procedure and for the tokens to be passed as CLOB as well from the stored procedure back to Oracle Text.

The user-defined lexer indexing procedure should use this interface when at least one of the documents in the column to be indexed is larger than 32512 bytes or the corresponding tokens are represented by more than 32512 bytes.
The first and second parameters are temporary CLOBs. Avoid assigning these CLOB locators to other locator variables. Assigning the formal parameter CLOB locator to another locator variable causes a new copy of the temporary CLOB to be created resulting in a performance hit.

**QUERY_PROCEDURE**

This callback stored procedure is called by Oracle Text as needed to tokenize *words* in the query. A space-delimited group of characters (excluding the query operators) in the query will be identified by Oracle Text as a *word*.

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

The index owner must have EXECUTE privilege on this stored procedure.
This stored procedure must not be replaced or be dropped after the index is created. You can replace or drop this stored procedure after the index is dropped.

**Restrictions** This procedure must not perform any of the following operations:
- rollback
- explicitly or implicitly commit the current transaction
- issue any other transaction control statement
- alter the session language or territory

The child elements of the root element tokens of the XML document returned must be in the same order as the tokens occur in the query word being tokenized.

The behavior of this stored procedure must be deterministic with respect to all parameters.

**Parameters** Table 2–4 describes the interface for the user-defined lexer query procedure:

<table>
<thead>
<tr>
<th>Parameter Position</th>
<th>Parameter Mode</th>
<th>Parameter Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IN</td>
<td>VARCHAR2</td>
<td>Query word to be tokenized.</td>
</tr>
<tr>
<td>2</td>
<td>IN</td>
<td>CTX_ULEXER_WILDCARD_TAB</td>
<td>Character offsets of wildcard characters (% and _) in the query word. If the query word passed in by Oracle Text does not contain any wildcard characters then this index-by table will be empty. The wildcard characters in the query word must be preserved in the tokens returned in order for the wildcard query feature to work properly. The character offset is 0 (zero) based.</td>
</tr>
<tr>
<td>3</td>
<td>IN OUT</td>
<td>VARCHAR2</td>
<td>Tokens encoded as XML. If the query word contains no tokens then either NULL must be returned or the tokens element in the XML document returned must contain no child elements. The length of the data must be less-than or equal to 32512 bytes.</td>
</tr>
</tbody>
</table>
Encoding Tokens as XML

The sequence of tokens returned by your stored procedure must be represented as an XML 1.0 document. The XML document must be valid with respect to the XML Schemas given in the following sections.

- XML Schema for No-Location, User-defined Indexing Procedure
- XML Schema for User-defined Indexing Procedure with Location
- XML Schema for User-defined Lexer Query Procedure

Limitations

To boost performance of this feature, the XML parser in Oracle Text will not perform validation and will not be a full-featured XML compliant parser. This implies that only minimal XML features will be supported. The following XML features are not supported:

- Document Type Declaration (for example, <!DOCTYPE [...]> ) and therefore entity declarations. Only the following built-in entities can be referenced: lt, gt, amp, quot, and apos.
- CDATA sections.
- Comments.
- Processing Instructions.
- XML declaration (for example, <?xml version="1.0" ...?>).
- Namespaces.
- Use of elements and attributes other than those defined by the corresponding XML Schema.
- Character references (for example &#x099F;).
- xml:space attribute.
- xml:lang attribute

XML Schema for No-Location, User-defined Indexing Procedure

This section describes additional constraints imposed on the XML document returned by the user-defined lexer indexing procedure when the third parameter is FALSE. The XML document returned must be valid with respect to the following XML Schema:

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="tokens">
```

2-62 Oracle Text Reference
Here are some of the constraints imposed by this XML Schema:

- The root element is tokens. This is mandatory. It has no attributes.
- The root element can have zero or more child elements. The child elements can be one of the following: eos, eop, num, word, and compMem. Each of these represent a specific type of token.
- The compMem element must be preceded by a word element or a compMem element.
- The eos and eop elements have no attributes and must be empty elements.
- The num, word, and compMem elements have no attributes. Oracle Text will normalize the content of these elements as follows: convert whitespace characters to space characters, collapse adjacent space characters to a single
space character, remove leading and trailing spaces, perform entity reference replacement, and truncate to 64 bytes.

Table 2–5 describes the element names defined in the preceding XML Schema.

**Table 2–5  Element names**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>This element represents a simple word token. The content of the element is the word itself. Oracle Text does the work of identifying this token as being a stop word or non-stop word and processing it appropriately.</td>
</tr>
<tr>
<td>num</td>
<td>This element represents an arithmetic number token. The content of the element is the arithmetic number itself. Oracle Text treats this token as a stop word if the stoplist preference has NUMBERS added as the stopclass. Otherwise this token is treated the same way as the word token. Supported this token type is optional. Without support for this token type, adding the NUMERBS stopclass will have no effect.</td>
</tr>
<tr>
<td>eos</td>
<td>This element represents end-of-sentence token. Oracle Text uses this information so that it can support WITHIN SENTENCE queries. Supported this token type is optional. Without support for this token type, queries against the SENTENCE section will not work as expected.</td>
</tr>
<tr>
<td>eop</td>
<td>This element represents end-of-paragraph token. Oracle Text uses this information so that it can support WITHIN PARAGRAPH queries. Supported this token type is optional. Without support for this token type, queries against the PARAGRAPH section will not work as expected.</td>
</tr>
<tr>
<td>compMem</td>
<td>Same as the word element, except that the implicit word offset is the same as the previous word token. Support for this token type is optional.</td>
</tr>
</tbody>
</table>

**Example** Document: Vom Nordhauptbahnhof und aus der Innenstadt zum Messegelände.

**Tokens:**

```xml
<tokens>
  <word> VOM </word>
  <word> NORDHAUPTBAHNHOF </word>
  <compMem>NORD</compMem>
  <compMem>HAUPT </compMem>
  <compMem>BAHNHOF </compMem>
  <compMem>HAUPTBAHNHOF </compMem>
  <word> UND </word>
</tokens>
```
Example Document: Oracle10g Release 1

Tokens:
<tokens>
  <word> ORACLE10G </word>
  <word> RELEASE </word>
  <num> 1 </num>
</tokens>

Example Document: WHERE salary < 25000.00 AND job = 'F&B Manager'

Tokens:
<tokens>
  <word> WHERE </word>
  <word> salary &lt; 25000.00 </word>
  <word> AND </word>
  <word> job </word>
  <word> F&amp;B </word>
  <word> Manager </word>
</tokens>

XML Schema for User-defined Indexing Procedure with Location

This section describes additional constraints imposed on the XML document returned by the user-defined lexer indexing procedure when the third parameter is TRUE. The XML document returned must be valid w.r.t the following XML schema:

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="tokens">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
Lexer Types

```xml
<xsd:element name="eos" type="EmptyTokenType"/>
<xsd:element name="eop" type="EmptyTokenType"/>
<xsd:element name="num" type="DocServiceTokenType"/>
<xsd:group ref="DocServiceCompositeGroup"/>
</xsd:choice>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<!-- Enforce constraint that compMem element must be preceded by word element or compMem element for document service -->
<xsd:group name="DocServiceCompositeGroup">
  <xsd:sequence>
    <xsd:element name="word" type="DocServiceTokenType"/>
    <xsd:element name="compMem" type="DocServiceTokenType" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>

<!-- EmptyTokenType defines an empty element without attributes -->
<xsd:complexType name="EmptyTokenType"/>

<!-- DocServiceTokenType defines an element with content and mandatory attributes -->
<xsd:complexType name="DocServiceTokenType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:token">
      <xsd:attribute name="off" type="OffsetType" use="required"/>
      <xsd:attribute name="len" type="xsd:unsignedShort" use="required"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="OffsetType">
  <xsd:restriction base="xsd:unsignedInt">
    <xsd:maxInclusive value="2147483647"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```
Some of the constraints imposed by this XML Schema are as follows:

- The root element is tokens. This is mandatory. It has no attributes.
- The root element can have zero or more child elements. The child elements can be one of the following: eos, eop, num, word, and compMem. Each of these represent a specific type of token.
- The compMem element must be preceded by a word element or a compMem element.
- The eos and eop elements have no attributes and must be empty elements.
- The num, word, and compMem elements have two mandatory attributes: off and len. Oracle Text will normalize the content of these elements as follows: convert whitespace characters to space characters, collapse adjacent space characters to a single space character, remove leading and trailing spaces, perform entity reference replacement, and truncate to 64 bytes.
- The off attribute value must be an integer between 0 and 2147483647 inclusive.
- The len attribute value must be an integer between 0 and 65535 inclusive.

Table 2–5, "Element names" describes the element types defined in the preceding XML Schema.

Table 2–6, "Attributes" describes the attributes defined in the preceding XML Schema.

### Table 2–6 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>This attribute represents the character offset of the token as it appears in the document being tokenized. The offset is with respect to the character document passed to the user-defined lexer indexing procedure, not the document fetched by the datastore. The document fetched by the datastore may be pre-processed by the filter object or the section group object, or both, before being passed to the user-defined lexer indexing procedure. The offset of the first character in the document being tokenized is 0 (zero).</td>
</tr>
</tbody>
</table>
Sum of `off` attribute value and `len` attribute value must be less than or equal to the total number of characters in the document being tokenized. This is to ensure that the document offset and characters being referenced are within the document boundary.

**Example**  Document: User-defined Lexer.

Tokens:

```xml
<tokens>
  <word off="0" len="4"> USE </word>
  <word off="5" len="7"> DEF </word>
  <word off="13" len="5"> LEX </word>
  <eos/>
</tokens>
```

**XML Schema for User-defined Lexer Query Procedure**

This section describes additional constraints imposed on the XML document returned by the user-defined lexer query procedure. The XML document returned must be valid with respect to the following XML Schema:

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="tokens">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="num" type="QueryTokenType"/>
          <xsd:element name="word" type="QueryTokenType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
Here are some of the constraints imposed by this XML Schema:

- The root element is tokens. This is mandatory. It has no attributes.
- The root element can have zero or more child elements. The child elements can be one of the following: num and word. Each of these represent a specific type of token.
- The compMem element must be preceded by a word element or a compMem element.
- The num and word elements have a single optional attribute: wildcard. Oracle Text will normalize the content of these elements as follows: convert whitespace characters to space characters, collapse adjacent space characters to a single
space character, remove leading and trailing spaces, perform entity reference replacement, and truncate to 64 bytes.

- The wildcard attribute value is a white-space separated list of integers. The minimum number of integers is 1 and the maximum number of integers is 64. The value of the integers must be between 0 and 378 inclusive. The intriguers in the list can be in any order.

Table 2–5, "Element names" describes the element types defined in the preceding XML Schema.

Table 2–7, "Attribute for XML Schema: Query Procedure" describes the attribute defined in the preceding XML Schema.

**Table 2–7  Attribute for XML Schema: Query Procedure**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wildcard</td>
<td>Any % or _ characters in the query which are not escaped by the user are considered wildcard characters because they are replaced by other characters. These wildcard characters in the query must be preserved during tokenization in order for the wildcard query feature to work properly. This attribute represents the character offsets (same semantics as SQL function LENGTH) of wildcard characters in the content of the element. Oracle Text will adjust these offsets for any normalization performed on the content of the element. The characters pointed to by the offsets must either be % or _ characters. The offset of the first character in the content of the element is 0. If the token does not contain any wildcard characters then this attribute must not be specified.</td>
</tr>
</tbody>
</table>

**Example** Query word: pseudo-%morph%

Tokens:

```xml
<tokens>
  <word> PSEUDO </word>
  <word wildcard="1 7"> %MORPH% </word>
</tokens>
```
**Example** Query word: `<%>
Tokens:
<tokens>
    <word wildcard="5"> &lt;%&gt; </word>
</tokens>

**WORLD_LEXER**

Use the `WORLD_LEXER` to index text columns that contain documents of different languages. For example, you can use this lexer to index a text column that stores English, Japanese, and German documents.

`WORLD_LEXER` differs from `MULTI_LEXER` in that `WORLD_LEXER` automatically detects the language(s) of a document. Unlike `MULTI_LEXER`, `WORLD_LEXER` does not require you to have a language column in your base table or to specify the language column when you create the index. Moreover, it is not necessary to use sub-lexers, as with `MULTI_LEXER`. (See `MULTI_LEXER` on page 2-46.)

However, many features that work with `MULTI_LEXER` do not work with `WORLD_LEXER`. For space-delimited language, these include ABOUT, Broader Term, Fuzzy, Narrower Term, Preferred Term, Related Term, soundex, stem, SYNonym, Translation Term, Translation Term Synonym, and Top Term. Additionally, for languages that are not space-delimited, EQUIValence and wildcards also do not work with `WORLD_LEXER`.

This lexer has no attributes.

`WORLD_LEXER` works with languages whose character sets are defined by the Unicode 4.0 standard. For a list of languages that `WORLD_LEXER` can work with, see "World Lexer Features" on page D-5.

**WORLD_LEXER Example**

Here is an example of creating an index using `WORLD_LEXER`.

```sql
exec ctx_ddl.create_preference('MYLEXER', 'world_lexer');
create index doc_idx on doc(data)
    indextype is CONTEXT
    parameters ('lexer MYLEXER
                stoplist CTXSYS.EMPTY_STOPLIST');
```
Wordlist Type

Use the wordlist preference to enable the query options such as stemming, fuzzy matching for your language. You can also use the wordlist preference to enable substring and prefix indexing which improves performance for wildcard queries with CONTAINS and CATSEARCH.

To create a wordlist preference, you must use BASIC_WORDLIST, which is the only type available.

BASIC_WORDLIST

Use BASIC_WORDLIST type to enable stemming and fuzzy matching or to create prefix indexes with Text indexes.

See Also: For more information about the stem and fuzzy operators, see Chapter 3, “CONTAINS Query Operators”.

BASIC_WORDLIST has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>stemmer</td>
<td>Specify which language stemmer to use. You can specify one of the following: NULL (no stemming) ENGLISH (English inflectional) DERIVATIONAL (English derivational) DUTCH FRENCH GERMAN ITALIAN SPANISH AUTO (Automatic language-detection for stemming for the languages above. Does not auto-detect Japanese.) JAPANESE</td>
</tr>
</tbody>
</table>
### Table 2–8  BASIC_WORDLIST Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>fuzzy_match</td>
<td>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)</td>
</tr>
<tr>
<td>fuzzy_score</td>
<td>Specify a default lower limit of fuzzy score. Specify a number between 0 and 80. Text with scores below this number is not returned. Default is 60.</td>
</tr>
<tr>
<td>fuzzy_numresults</td>
<td>Specify the maximum number of fuzzy expansions. Use a number between 0 and 5,000. Default is 100.</td>
</tr>
<tr>
<td>substring_index</td>
<td>Specify TRUE for Oracle Text to create a substring index. A substring index improves left-truncated and double-truncated wildcard queries such as %ing or %benz%. Default is FALSE.</td>
</tr>
<tr>
<td>prefix_index</td>
<td>Specify TRUE to enable prefix indexing. Prefix indexing improves performance for right truncated wildcard searches such as TO%. Defaults to FALSE.</td>
</tr>
<tr>
<td>prefix_length_min</td>
<td>Specify the minimum length of indexed prefixes. Defaults to 1.</td>
</tr>
<tr>
<td>prefix_length_max</td>
<td>Specify the maximum length of indexed prefixes. Defaults to 64.</td>
</tr>
</tbody>
</table>
stemmer

Specify the stemmer used for word stemming in Text queries. When you do not specify a value for stemmer, the default is ENGLISH.

Specify AUTO for the system to automatically set the stemming language according to the language setting of the session. When there is no stemmer for a language, the default is NULL. With the NULL stemmer, the stem operator is ignored in queries.

You can create your own stemming user-dictionary. See “Stemming User-Dictionaries” on page 2-42 for more information.

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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>wildcard_maxterms</td>
<td>Specify the maximum number of terms in a wildcard expansion. Use a number between 1 and 15,000. Default is 5,000.</td>
</tr>
</tbody>
</table>

Note: The fuzzy_match attribute values for Chinese and Korean are dummy attribute values that prevent the English and Japanese fuzzy matching routines from being used on Chinese and Korean text.

The default for fuzzy_match is GENERIC.

Specify AUTO for the system to automatically set the fuzzy matching language according to language setting of the session.

fuzzy_score

Specify a default lower limit of fuzzy score. Specify a number between 0 and 80. Text with scores below this number are not returned. The default is 60.

Fuzzy score is a measure of how close the expanded word is to the query word. The higher the score the better the match. Use this parameter to limit fuzzy expansions to the best matches.
fuzzy_numresults
Specify the maximum number of fuzzy expansions. Use a number between 0 and 5000. The default is 100.

Setting a fuzzy expansion limits the expansion to a specified number of the best matching words.

substring_index
Specify TRUE for Oracle Text to create a substring index. A substring index improves performance for left-truncated or double-truncated wildcard queries such as %ing or %benz%. The default is false.

Substring indexing has the following impact on indexing and disk resources:

- Index creation and DML processing is up to 4 times slower
- The size of the substring index created is approximately the size of the $X index on the word table.
- Index creation with substring_index enabled requires more rollback segments during index flushes than with substring index off. Oracle recommends that you do either of the following when creating a substring index:
  - make available double the usual rollback
  - decrease the index memory to reduce the size of the index flushes to disk

prefix_index
Specify yes to enable prefix indexing. Prefix indexing improves performance for right truncated wildcard searches such as TO%. Defaults to NO.

Note: Enabling prefix indexing increases index size.

Prefix indexing chops up tokens into multiple prefixes to store in the $I table. For example, words TOKEN and TOY are normally indexed like this in the $I table:

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>0</td>
<td>DOCID 1 POS 1</td>
</tr>
<tr>
<td>TOY</td>
<td>0</td>
<td>DOCID 1 POS 3</td>
</tr>
</tbody>
</table>
With prefix indexing, Oracle Text indexes the prefix substrings of these tokens as follows with a new token type of 6:

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>0</td>
<td>DOCID 1 POS 1</td>
</tr>
<tr>
<td>TOY</td>
<td>0</td>
<td>DOCID 1 POS 3</td>
</tr>
<tr>
<td>T</td>
<td>6</td>
<td>DOCID 1 POS 1 POS 3</td>
</tr>
<tr>
<td>TO</td>
<td>6</td>
<td>DOCID 1 POS 1 POS 3</td>
</tr>
<tr>
<td>TOK</td>
<td>6</td>
<td>DOCID 1 POS 1</td>
</tr>
<tr>
<td>TOKE</td>
<td>6</td>
<td>DOCID 1 POS 1</td>
</tr>
<tr>
<td>TOKEN</td>
<td>6</td>
<td>DOCID 1 POS 1</td>
</tr>
<tr>
<td>TOY</td>
<td>6</td>
<td>DOCID 1 POS 3</td>
</tr>
</tbody>
</table>

Wildcard searches such as TO% are now faster because Oracle Text does no expansion of terms and merging of result sets. To obtain the result, Oracle Text need only examine the (TO,6) row.

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

For example, setting `prefix_length_min` to 3 and `prefix_length_max` to 5 indexes all prefixes between 3 and 5 characters long.

---

**Note:** A wildcard search whose pattern is below the minimum length or above the maximum length is searched using the slower method of equivalence expansion and merging.

**prefix_length_max**
Specify the maximum length of indexed prefixes. Defaults to 64.

For example, setting `prefix_length_min` to 3 and `prefix_length_max` to 5 indexes all prefixes between 3 and 5 characters long.

---

**Note:** A wildcard search whose pattern is below the minimum length or above the maximum length is searched using the slower method of equivalence expansion and merging.
**wildcard_maxterms**
Specify the maximum number of terms in a wildcard (%) expansion. Use this parameter to keep wildcard query performance within an acceptable limit. Oracle Text returns an error when the wildcard query expansion exceeds this number.

**BASIC_WORDLIST Example**
The following example shows the use of the BASIC_WORDLIST type.

**Enabling Fuzzy Matching and Stemming**
The following example enables stemming and fuzzy matching for English. The preference STEM_FUZZY_PREF sets the number of expansions to the maximum allowed. This preference also instructs the system to create a substring index to improve the performance of double-truncated searches.

```sql
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:

```sql
create index fuzzy_stem_subst_idx on mytable ( docs )
    indextype is ctxsys.context parameters ('Wordlist STEM_FUZZY_PREF');
```

**Enabling Sub-string and Prefix Indexing**
The following example sets the wordlist preference for prefix and sub-string indexing. For prefix indexing, it specifies that Oracle Text create token prefixes between 3 and 4 characters long:

```sql
begin
    ctx_ddl.create_preference('mywordlist', 'BASIC_WORDLIST');
    ctx_ddl.set_attribute('mywordlist','PREFIX_INDEX','TRUE');
    ctx_ddl.set_attribute('mywordlist','PREFIX_MIN_LENGTH',3);
    ctx_ddl.set_attribute('mywordlist','PREFIX_MAX_LENGTH',4);
    ctx_ddl.set_attribute('mywordlist','SUBSTRING_INDEX', 'YES');
end
```
Setting Wildcard Expansion Limit

Use the wildcard_maxterms attribute to set the maximum allowed terms in a wildcard expansion.

```sql
--- create a sample table
drop table quick;
create table quick
(
    quick_id number primary key,
    text      varchar(80)
);

--- insert a row with 10 expansions for 'tire%'
insert into quick ( quick_id, text )
    values ( 1, 'tire tirea tireb tirec tired tiree tiref tireg tireh tirei
tirej' ) ;
commit;

--- create an index using wildcard_maxterms=100
begin
    Ctx_Ddl.Create_Preference('wildcard_pref', 'BASIC_WORDLIST');
    ctx_ddl.set_attribute('wildcard_pref', 'wildcard_maxterms', 100) ;
end;
/
create index wildcard_idx on quick(text)
    indextype is ctxsys.context
    parameters ('Wordlist wildcard_pref');

--- query on 'tire%' - should work fine
select quick_id from quick
    where contains ( text, 'tire%' ) > 0;

--- now re-create the index with wildcard_maxterms=5

drop index wildcard_idx ;

begin
    Ctx_Ddl.Drop_Preference('wildcard_pref');
    Ctx_Ddl.Create_Preference('wildcard_pref', 'BASIC_WORDLIST');
    ctx_ddl.set_attribute('wildcard_pref', 'wildcard_maxterms', 5) ;
end;
/
create index wildcard_idx on quick(text)
    indextype is ctxsys.context
```
parameters ('Wordlist wildcard_pref')

--- query on 'tire%' gives "wildcard query expansion resulted in too many terms"
select quick_id from quick
where contains (text, 'tire%') > 0;

Storage Types

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

<table>
<thead>
<tr>
<th>type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASIC_STORAGE</td>
<td>Indexing type used to specify the tablespace and creation parameters for</td>
</tr>
<tr>
<td></td>
<td>the database tables and indexes that constitute a Text index.</td>
</tr>
</tbody>
</table>

**BASIC_STORAGE**

The BASIC_STORAGE type specifies the tablespace and creation parameters for the database tables and indexes that constitute a Text index.

The clause you specify is added to the internal CREATE TABLE (CREATE INDEX for the i_index_clause) statement at index creation. You can specify most allowable clauses, such as storage, LOB storage, or partitioning. However, you cannot specify an index organized table clause.

**See Also:** For more information about how to specify CREATE TABLE and CREATE INDEX statements, see Oracle Database SQL Reference.

BASIC_STORAGE has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_table_clause</td>
<td>Parameter clause for dr$indexname$I table creation. Specify storage and</td>
</tr>
<tr>
<td></td>
<td>tablespace clauses to add to the end of the internal CREATE TABLE statement.</td>
</tr>
<tr>
<td></td>
<td>The I table is the index data table.</td>
</tr>
</tbody>
</table>
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:

```sql
create index IOWNER.idx on TOWNER.tab(b) indextype is ctxsys.context;
```

In this example, the text index is created in IOWNER's default tablespace.
Storage Example
The following examples specify that the index tables are to be created in the foo tablespace with an initial extent of 1K:

begin
ctx_ddl.create_preference('mystore', 'BASIC_STORAGE');
ctx_ddl.set_attribute('mystore', 'I_TABLE_CLAUSE',
    'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'K_TABLE_CLAUSE',
    'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'R_TABLE_CLAUSE',
    'tablespace users storage (initial 1K) lob
    (data) store as (disable storage in row cache)');
ctx_ddl.set_attribute('mystore', 'N_TABLE_CLAUSE',
    'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'I_INDEX_CLAUSE',
    'tablespace foo storage (initial 1K) compress 2');
ctx_ddl.set_attribute('mystore', 'P_TABLE_CLAUSE',
    'tablespace foo storage (initial 1K)');
end;

Section Group Types
In order to issue WITHIN queries on document sections, you must create a section group before you define your sections. You specify your section group in the parameter clause of CREATE INDEX.

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

<table>
<thead>
<tr>
<th>Section Group Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL_SECTION_GROUP</td>
<td>Use this group type when you define no sections or when you define only SENTENCE or PARAGRAPH sections. This is the default.</td>
</tr>
<tr>
<td>BASIC_SECTION_GROUP</td>
<td>Use this group type for defining sections where the start and end tags are of the form &lt;A&gt; and &lt;/A&gt;. Note: This group type does not support input such as unbalanced parentheses, comments tags, and attributes. Use HTML_SECTION_GROUP for this type of input.</td>
</tr>
<tr>
<td>HTML_SECTION_GROUP</td>
<td>Use this group type for indexing HTML documents and for defining sections in HTML documents.</td>
</tr>
</tbody>
</table>
Section Group Types

<table>
<thead>
<tr>
<th>Section Group Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML_SECTION_GROUP</td>
<td>Use this group type for indexing XML documents and for defining sections in XML documents. All sections to be indexed must be manually defined for this group.</td>
</tr>
<tr>
<td>AUTO_SECTION_GROUP</td>
<td>Use this group type to automatically create a zone section for each start-tag/end-tag pair in an XML document. The section names derived from XML tags are case sensitive as in XML. Attribute sections are created automatically for XML tags that have attributes. Attribute sections are named in the form tag@attribute. Stop sections, empty tags, processing instructions, and comments are not indexed. The following limitations apply to automatic section groups:</td>
</tr>
<tr>
<td></td>
<td>- You cannot add zone, field, or special sections to an automatic section group.</td>
</tr>
<tr>
<td></td>
<td>- You can define a stop section that applies only to one particular type; that is, if you have two different XML DTDs, both of which use a tag called FOO, you can define TYPE1 FOO to be stopped, but TYPE2 FOO to not be stopped.</td>
</tr>
<tr>
<td></td>
<td>- The length of the indexed tags, including prefix and namespace, cannot exceed 64 characters. Tags longer than this are not indexed.</td>
</tr>
<tr>
<td>PATH_SECTION_GROUP</td>
<td>Use this group type to index XML documents. Behaves like the AUTO_SECTION_GROUP. The difference is that with this section group you can do path searching with the INPATH and HASPATH operators. Queries are also case-sensitive for tag and attribute names. Stop sections are not allowed.</td>
</tr>
<tr>
<td>NEWS_SECTION_GROUP</td>
<td>Use this group for defining sections in newsgroup formatted documents according to RFC 1036.</td>
</tr>
</tbody>
</table>

Section Group Examples

This example shows the use of section groups in both HTML and XML documents.
Creating Section Groups in HTML Documents
The following statement creates a section group called htmgroup with the HTML group type.

```sql
begin
    ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
end;
```

You can optionally add sections to this group using the procedures in the CTX_DDL package, such as CTX_DDL.ADD_SPECIAL_SECTION or CTX_DDL.ADD_ZONE_SECTION. To index your documents, you can issue a statement such as:

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

See Also: For more information on section groups, see Chapter 7, "CTX_DDL Package"

Creating Sections Groups in XML Documents
The following statement creates a section group called xmlgroup with the XML_SECTION_GROUP group type.

```sql
begin
    ctx_ddl.create_section_group('xmlgroup', 'XML_SECTION_GROUP');
end;
```

You can optionally add sections to this group using the procedures in the CTX_DDL package, such as CTX_DDL.ADD_ATTR_SECTION or CTX_DDL.ADD_STOP_SECTION. To index your documents, you can issue a statement such as:

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

See Also: For more information on section groups, see Chapter 7, "CTX_DDL Package"

Automatic Sectioning in XML Documents
The following statement creates a section group called auto with the AUTO_SECTION_GROUP group type. This section group automatically creates sections from tags in XML documents.

```sql
begin
    ctx_ddl.create_section_group('auto', 'AUTO_SECTION_GROUP');
end;
```
 Classifier Types

This section describes the classifier types used to create a preference for CTX_CLS.TRAIN and CTXRULE index creation. The following two classifier types are supported:

- **RULE_CLASSIFIER**
- **SVM_CLASSIFIER**

**RULE_CLASSIFIER**

Use the RULE_CLASSIFIER type for creating preferences for the query rule generating procedure, CTX_CLS.TRAIN and for CTXRULE creation. The rules generated with this type are essentially query strings and can be easily examined. The queries generated by this classifier can use the AND, NOT, or ABOUT operators. The WITHIN operator is supported for queries on field sections only.

This type has the following attributes:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Default</th>
<th>Min Value</th>
<th>Max Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRESHOLD</td>
<td>I</td>
<td>50</td>
<td>1</td>
<td>99</td>
<td>Specify threshold (in percentage) for rule generation. One rule is output only when its confidence level is larger than threshold.</td>
</tr>
<tr>
<td>MAX_TERMS</td>
<td>I</td>
<td>100</td>
<td>20</td>
<td>2000</td>
<td>For each class, a list of relevant terms is selected to form rules. Specify the maximum number of terms that can be selected for each class.</td>
</tr>
<tr>
<td>MEMORY_SIZE</td>
<td>I</td>
<td>500</td>
<td>10</td>
<td>4000</td>
<td>Specify memory usage for training in MB. Larger values improve performance.</td>
</tr>
</tbody>
</table>
Classifer Types

SVM_CLASSIFIER

Use the SVM_CLASSIFIER type for creating preferences for the rule generating procedure, CTX_CLS.TRAIN, and for CTXRULE creation. This classifier type represents the Support Vector Machine method of classification and generates rules in binary format. Use this classifier type when you need high classification accuracy.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Default</th>
<th>Min Value</th>
<th>Max Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT_THRESHOLD</td>
<td>F</td>
<td>0.001</td>
<td>0</td>
<td>0.90</td>
<td>Specify a threshold for term selection. There are two thresholds guiding two steps in selecting relevant terms. This threshold controls the behavior of the first step. At this step, terms are selected as candidate terms for the further consideration in the second step. The term is chosen when the ratio of the occurrence frequency over the number of documents in the training set is larger than this threshold.</td>
</tr>
<tr>
<td>TERM_THRESHOLD</td>
<td>I</td>
<td>10</td>
<td>0</td>
<td>100</td>
<td>Specify a threshold as a percentage for term selection. This threshold controls the second step term selection. Each candidate term has a numerical quantity calculated to imply its correlation with a given class. The candidate term will be selected for this class only when the ratio of its quantity value over the maximum value for all candidate terms in the class is larger than this threshold.</td>
</tr>
<tr>
<td>PRUNE_LEVEL</td>
<td>I</td>
<td>75</td>
<td>0</td>
<td>100</td>
<td>Specify how much to prune a built decision tree for better coverage. Higher values mean more aggressive pruning and the generated rules will have larger coverage but less accuracy.</td>
</tr>
</tbody>
</table>
This type has the following attributes:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Default</th>
<th>Min Value</th>
<th>Max Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_DOCTERMS</td>
<td>I</td>
<td>50</td>
<td>10</td>
<td>8192</td>
<td>Specify the maximum number of terms representing one document.</td>
</tr>
<tr>
<td>MAX_FEATURES</td>
<td>I</td>
<td>3,000</td>
<td>1</td>
<td>100,000</td>
<td>Specify the maximum number of distinct features.</td>
</tr>
<tr>
<td>THEME_ON</td>
<td>B</td>
<td>FALSE</td>
<td>NULL</td>
<td>NULL</td>
<td>Specify TRUE to use themes as features.</td>
</tr>
<tr>
<td>TOKEN_ON</td>
<td>B</td>
<td>TRUE</td>
<td>NULL</td>
<td>NULL</td>
<td>Specify TRUE to use regular tokens as features.</td>
</tr>
<tr>
<td>STEM_ON</td>
<td>B</td>
<td>FALSE</td>
<td>NULL</td>
<td>NULL</td>
<td>Specify TRUE to use stemmed tokens as features. Only works when turning INDEX_STEM on for the lexer.</td>
</tr>
<tr>
<td>MEMORY_SIZE</td>
<td>I</td>
<td>500</td>
<td>10</td>
<td>4000</td>
<td>Specify approximate memory size in MB.</td>
</tr>
<tr>
<td>SECTION_WEIGHT</td>
<td>I</td>
<td>2</td>
<td>0</td>
<td>100</td>
<td>Specify the occurrence multiplier for adding a term in a field section as a normal term. For example, by default, the term cat in &quot;&lt;A&gt;cat&lt;/A&gt;&quot; is a field section term and is treated as a normal term with occurrence equal to 2, but you can specify that it be treated as a normal term with a weight up to 100. SECTION_WEIGHT is only meaningful when the index policy specifies a field section.</td>
</tr>
</tbody>
</table>

**Cluster Types**

This section describes the cluster types used for creating preferences for the CTX_CLS.CLUSTERING procedure.
KMEAN_CLUSTER

This clustering type has the following attributes:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Default</th>
<th>Min Value</th>
<th>Max Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_DOCTERMS</td>
<td>I</td>
<td>50</td>
<td>10</td>
<td>8192</td>
<td>Specify the maximum number of distinct terms representing one document.</td>
</tr>
<tr>
<td>MAX_FEATURES</td>
<td>I</td>
<td>3,000</td>
<td>1</td>
<td>500,000</td>
<td>Specify the maximum number of distinct features.</td>
</tr>
<tr>
<td>THEME_ON</td>
<td>B</td>
<td>FALSE</td>
<td>NULL</td>
<td>NULL</td>
<td>Specify TRUE to use themes as features.</td>
</tr>
<tr>
<td>TOKEN_ON</td>
<td>B</td>
<td>TRUE</td>
<td>NULL</td>
<td>NULL</td>
<td>Specify TRUE to use regular tokens as features.</td>
</tr>
<tr>
<td>STEM_ON</td>
<td>B</td>
<td>FALSE</td>
<td>NULL</td>
<td>NULL</td>
<td>Specify TRUE to use stemmed tokens as features. This only works when turning INDEX_STEM on for the lexer.</td>
</tr>
<tr>
<td>MEMORY_SIZE</td>
<td>I</td>
<td>500</td>
<td>10</td>
<td>4000</td>
<td>Specify approximate memory size in MB.</td>
</tr>
<tr>
<td>SECTION_WEIGHT</td>
<td>I</td>
<td>2</td>
<td>0</td>
<td>100</td>
<td>Specify the occurrence multiplier for adding a term in a field section as a normal term. For example, by default, the term cat in &quot;&lt;A&gt;cat&lt;/A&gt;&quot; is a field section term and is treated as a normal term with occurrence equal to 2, but you can specify that it be treated as a normal term with a weight up to 100. SECTION_WEIGHT is only meaningful when the index policy specifies a field section.</td>
</tr>
<tr>
<td>CLUSTER_NUM</td>
<td>I</td>
<td>200</td>
<td>2</td>
<td>20000</td>
<td>Specify the maximum number of clusters to be generated. See the Hierarchical Clustering section that follows.</td>
</tr>
</tbody>
</table>
Hierarchical Clustering

If the HIERARCHY_DEPTH attribute is greater than 1, Oracle Text produces a hierarchy of clusters, in which one cluster is considered a child of another. For example, a cluster that contains documents about dogs might be the child of a cluster about animals. Producing a hierarchical cluster affords greater refinement of clustering; however, it can result in a performance hit.

The effect of the CLUSTER_NUM and MIN_SIMILARITY attributes depends on whether hierarchical clustering is selected or not.

In non-hierarchical clustering, CLUSTER_NUM refers to the total or maximum number of clusters to produce.

In hierarchical clustering, CLUSTER_NUM refers to the total or maximum number of clusters produced by the partitioning of a given cluster node. Since many nodes may split, a hierarchy layer can contain many more nodes than the value of CLUSTER_NUM.

The following table gives an example of how setting various attributes works for both hierarchical and non-hierarchical clustering, if CLUSTER_NUM is set to five.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Default</th>
<th>Min Value</th>
<th>Max Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN_SIMILARITY</td>
<td>F</td>
<td>0.2</td>
<td>0.01</td>
<td>0.99</td>
<td>Specify the minimum similarity score for each cluster (leaf cluster). There is no effect when hierarchical clustering is not used. See the Hierarchical Clustering section that follows.</td>
</tr>
<tr>
<td>HIERARCHY_DEPTH</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>20</td>
<td>The maximum depth of hierarchy. See the Hierarchical Clustering section that follows.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HIERARCHYDEPTH</th>
<th>CLUSTER_NUM</th>
<th>MIN_SIMILARITY</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>any</td>
<td>5 clusters total; no hierarchy</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0.2</td>
<td>Up to 5 child clusters produced for each parent cluster node. The hierarchy depth is about 2 (it may be larger than 2)</td>
</tr>
</tbody>
</table>
Stoplists

Stoplists identify the words in your language that are not to be indexed. In English, you can also identify stopthemes that are not to be indexed. By default, the system indexes text using the system-supplied stoplist that corresponds to your database language.

Oracle Text provides default stoplists for most common languages including English, French, German, Spanish, Dutch, and Danish. These default stoplists contain only stopwords.

See Also: For more information about the supplied default stoplists, see Appendix E, "Supplied Stoplists".

Multi-Language Stoplists

You can create multi-language stoplists to hold language-specific stopwords. A multi-language stoplist is useful when you use the MULTI_LEXER to index a table that contains documents in different languages, such as English, German, and Japanese.

To create a multi-language stoplist, use the CTX_DLL.CREATE_STOPLIST procedure and specify a stoplist type of MULTI_STOPLIST. You add language specific stopwords with CTX_DDL.ADD_STOPWORD.

At indexing time, the language column of each document is examined, and only the stopwords for that language are eliminated. At query time, the session language setting determines the active stopwords, like it determines the active lexer when using the multi-lexer.

Creating Stoplists

You can create your own stoplists using the CTX_DLL.CREATE_STOPLIST procedure. With this procedure you can create a BASIC_STOPLIST for single language stoplist, or you can create a MULTI_STOPLIST for a multi-language stoplist.

When you create your own stoplist, you must specify it in the parameter clause of CREATE INDEX.
Modifying the Default Stoplist

The default stoplist is always named `CTXSYS.DEFAULT_STOPLIST`. You can use the following procedures to modify this stoplist:

- `CTX_DDL.ADD_STOPWORD`
- `CTX_DDL.REMOVE_STOPWORD`
- `CTX_DDL.ADD_STOPTHEME`
- `CTX_DDL.ADD_STOPCLASS`

When you modify `CTXSYS.DEFAULT_STOPLIST` with the `CTX_DDL` package, you must re-create your index for the changes to take effect.

Dynamic Addition of Stopwords

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

**Note:** Even though you can dynamically add stopwords to an index, you cannot dynamically remove stopwords. To remove a stopword, you must use `CTX_DDL.REMOVE_STOPWORD`, drop your index and re-create it.

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

System-Defined Preferences

When you install Oracle Text, some indexing preferences are created. You can use these preferences in the parameter clause of `CREATE INDEX` or define your own.

The default index parameters are mapped to some of the system-defined preferences described in this section.

**See Also:** For more information about default index parameters, see "Default Index Parameters" on page 2-95.

System-defined preferences are divided into the following categories:

- Data Storage
Data Storage

This section discusses the types associated with data storage preferences.

**CTXSYS.DEFAULT_DATASTORE**
This preference uses the **DIRECT_DATASTORE** type. You can use this preference to create indexes for text columns in which the text is stored directly in the column.

**CTXSYS.FILE_DATASTORE**
This preference uses the **FILE_DATASTORE** type.

**CTXSYS.URL_DATASTORE**
This preference uses the **URL_DATASTORE** type.

Filter

This section discusses the types associated with filtering preferences.

**CTXSYS.NULL_FILTER**
This preference uses the **NULL_FILTER** type.

**CTXSYS.INSO_FILTER**
This preference uses the **INSO_FILTER** type.

Lexer

This section discusses the types associated with lexer preferences.
CTXSYS.DEFAULT_LEXER
The default lexer depends on the language used at install time. The following sections describe the default settings for CTXSYS.DEFAULT_LEXER for each language.

American and English Language Settings  If your language is English, this preference uses the BASIC_LEXER with the index_themes attribute disabled.

Danish Language Settings  If your language is Danish, this preference uses the BASIC_LEXER with the following option enabled:
- alternate spelling (alternate_spelling attribute set to DANISH)

Dutch Language Settings  If your language is Dutch, this preference uses the BASIC_LEXER with the following options enabled:
- composite indexing (composite attribute set to DUTCH)

German and German DIN Language Settings  If your language is German, 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 your language is Japanese, this preference uses the JAPANESE_VGRAM_LEXER.

Korean Language Settings  If your language is Korean, this preference uses the KOREAN_MORPH_LEXER. All attributes for the KOREAN_MORPH_LEXER are enabled.

Chinese Language Settings  If your language is Simplified or Traditional Chinese, this preference uses the CHINESE_VGRAM_LEXER.
Other Languages  For all other languages not listed in this section, this preference uses the BASIC_LEXER with no attributes set.

See Also:  To learn more about these options, see BASIC_LEXER on page 2-36.

CTXSYS.BASIC_LEXER
This preference uses the BASIC_LEXER.

Section Group
This section discusses the types associated with section group preferences.

CTXSYS.NULL_SECTION_GROUP
This preference uses the NULL_SECTION_GROUP type.

CTXSYS.HTML_SECTION_GROUP
This preference uses the HTML_SECTION_GROUP type.

CTXSYS.AUTO_SECTION_GROUP
This preference uses the AUTO_SECTION_GROUP type.

CTXSYS.PATH_SECTION_GROUP
This preference uses the PATH_SECTION_GROUP type.

Stoplist
This section discusses the types associated with stoplist preferences.

CTXSYS.DEFAULT_STOPLIST
This stoplist preference defaults to the stoplist of your database language.

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

CTXSYS.EMPTY_STOPLIST
This stoplist has no words.
Storage

This section discusses the types associated with storage preferences.

**CTXSYS.DEFAULT_STORAGE**
This storage preference uses the `BASIC_STORAGE` type.

Wordlist

This section discusses the types associated with wordlist preferences.

**CTXSYS.DEFAULT_WORDLIST**
This preference uses the language stemmer for your database language. If your language is not listed in Table 2–8 on page 2-72, this preference defaults to the NULL stemmer and the GENERIC fuzzy matching attribute.

System Parameters

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

- General System Parameters
- Default Index Parameters

General System Parameters

When you install Oracle Text, in addition to the system-defined preferences, the following system parameters are set:

<table>
<thead>
<tr>
<th>System Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_INDEX_MEMORY</td>
<td>This is the maximum indexing memory that can be specified in the parameter clause of <code>CREATE INDEX</code> and <code>ALTER INDEX</code>.</td>
</tr>
<tr>
<td>DEFAULT_INDEX_MEMORY</td>
<td>This is the default indexing memory used with <code>CREATE INDEX</code> and <code>ALTER INDEX</code>.</td>
</tr>
<tr>
<td>LOG_DIRECTORY</td>
<td>This is the directory for <code>CTX_OUTPUT</code> log files.</td>
</tr>
</tbody>
</table>
You can view system defaults by querying the CTX_PARAMETERS view. You can change defaults using the CTX_ADM.SET_PARAMETER procedure.

**Default Index Parameters**

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

**CONTEXT Index Parameters**

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

<table>
<thead>
<tr>
<th>System Parameter</th>
<th>Used When</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTX_DOC_KEY_TYPE</td>
<td>This is the default input key type, either ROWID or PRIMARY_KEY, for the CTX_DOC procedures. Set to ROWID at install time.</td>
<td>See also: CTX_DOC. SET_KEY_TYPE on page 8-37.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**System Parameter Description**

- **DEFAULT_DATASTORE**: No datastore preference specified in parameter clause of CREATE INDEX.
  - CTXSYS.DEFAULT_DATASTORE

- **DEFAULT_FILTER_FILE**: No filter preference specified in parameter clause of CREATE INDEX, and either of the following conditions is true:
  - Your files are stored in external files (BFILES) or
  - You specify a datastore preference that uses FILE_DATASTORE
  - CTXSYS.INSO_FILTER

- **DEFAULT_FILTER_BINARY**: No filter preference specified in parameter clause of CREATE INDEX, and Oracle Text detects that the text column datatype is RAW, LONG RAW, or BLOB.
  - CTXSYS.INSO_FILTER
## System Parameters

<table>
<thead>
<tr>
<th>System Parameter</th>
<th>Used When</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_FILTER_TEXT</td>
<td>No filter preference specified in parameter clause of CREATE INDEX, and Oracle Text detects that the text column datatype is either LONG, VARCHAR2, VARCHAR, CHAR, or CLOB.</td>
<td>CTXSYS.NULL_FILTER</td>
</tr>
</tbody>
</table>
| DEFAULT_SECTION_HTML          | No section group specified in parameter clause of CREATE INDEX, and when either of the following conditions is true:  
  - Your datastore preference uses URL_DATASTORE or  
  - Your filter preference uses INSO_FILTER. | CTXSYS.HTML_SECTION_GROUP   |
| DEFAULT_SECTION_TEXT          | No section group specified in parameter clause of CREATE INDEX, and when you do not use either URL_DATASTORE or INSO_FILTER. | CTXSYS.NULL_SECTION_GROUP   |
| DEFAULT_STORAGE               | No storage preference specified in parameter clause of CREATE INDEX.      | CTXSYS.DEFAULT_STORAGE      |
| DEFAULT_LEXER                 | No lexer preference specified in parameter clause of CREATE INDEX.        | CTXSYS.DEFAULT_LEXER        |
| DEFAULT_STOPLIST              | No stoplist specified in parameter clause of CREATE INDEX.                | CTXSYS.DEFAULT_STOPLIST     |
| DEFAULT_WORDLIST              | No wordlist preference specified in parameter clause of CREATE INDEX.     | CTXSYS.DEFAULT_WORDLIST     |

## CTXCAT Index Parameters

The following default parameters are used when you create a CTXCAT index with CREATE INDEX and do not specify any parameters in the parameter string. The CTXCAT index supports only the index set, lexer, storage, stoplist, and wordlist parameters. Each default parameter names a system-defined preference.

<table>
<thead>
<tr>
<th>System Parameter</th>
<th>Used When</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_CTXCAT_INDEX_SET</td>
<td>No index set specified in parameter clause of CREATE INDEX.</td>
<td>CTXSYS.DEFAULT_CTXCAT_INDEX_SET</td>
</tr>
<tr>
<td>DEFAULT_CTXCAT_STORAGE</td>
<td>No storage preference specified in parameter clause of CREATE INDEX.</td>
<td>CTXSYS.DEFAULT_CTXCAT_STORAGE</td>
</tr>
</tbody>
</table>
### CTXRULE Index Parameters

The following default parameters are used when you create a CTXRULE index with CREATE INDEX and do not specify any parameters in the parameter string. The CTXRULE index supports only the lexer, storage, stoplist, and wordlist parameters. Each default parameter names a system-defined preference.

<table>
<thead>
<tr>
<th>System Parameter</th>
<th>Used When</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_CTXRULE_LEXER</td>
<td>No lexer preference specified in parameter clause of CREATE INDEX.</td>
<td>CTXSYS.DEFAULT_LEXER</td>
</tr>
<tr>
<td>DEFAULT_CTXRULE_STORAGE</td>
<td>No storage preference specified in parameter clause of CREATE INDEX.</td>
<td>CTXSYS.DEFAULT_STORAGE</td>
</tr>
<tr>
<td>DEFAULT_CTXRULE_STOPLIST</td>
<td>No stoplist specified in parameter clause of CREATE INDEX.</td>
<td>CTXSYS.DEFAULT_STOPLIST</td>
</tr>
<tr>
<td>DEFAULT_CTXRULE_WORDLIST</td>
<td>No wordlist preference specified in parameter clause of CREATE INDEX.</td>
<td>CTXSYS.DEFAULT_WORDLIST</td>
</tr>
</tbody>
</table>

### Viewing Default Values

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

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

**Changing Default Values**
You can change a default value using the CTX_ADM.SET_PARAMETER procedure to name another custom or system-defined preference to use as default.
This chapter describes operator precedence and provides description, syntax, and examples for every CONTAINS operator. The following topics are covered:

- Operator Precedence
- ABOUT
- ACCUMulate (,)
- AND (&)
- Broader Term (BT, BTG, BTP, BTI)
- EQUIValence (=)
- Fuzzy
- HASPATH
- INPATH
- MDATA
- MINUS (-)
- Narrower Term (NT, NTG, NTP, NTI)
- NEAR ()
- NOT (~)
- OR ()
- Preferred Term (PT)
- Related Term (RT)
- soundex (!)
Operator Precedence

- stem ($)  
- Stored Query Expression (SQE)  
- SYNonym (SYN)  
- threshold (>)
- Translation Term (TR)  
- Translation Term Synonym (TRSYN)  
- Top Term (TT)  
- weight (*)  
- wildcards (% _)  
- WITHIN

Operator Precedence

Operator precedence determines the order in which the components of a query expression are evaluated. Text query operators can be divided into two sets of operators that have their own order of evaluation. These two groups are described later as Group 1 and Group 2.

In all cases, query expressions are evaluated in order from left to right according to the precedence of their operators. Operators with higher precedence are applied first. Operators of equal precedence are applied in order of their appearance in the expression from left to right.

Group 1 Operators

Within query expressions, the Group 1 operators have the following order of evaluation from highest precedence to lowest:

1. EQUIValence (=)
2. NEAR ()
3. weight (*), threshold (>)
4. MINUS (-)
5. NOT (~)
6. 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. stem ($)  
3. Fuzzy
4. soundex (!)

**Procedural Operators**

Other operators not listed under Group 1 or Group 2 are procedural. These operators have no sense of precedence attached to them. They include the SQE and thesaurus operators.

**Precedence Examples**

<table>
<thead>
<tr>
<th>Query Expression</th>
<th>Order of Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>w1</td>
<td>w2 &amp; w3</td>
</tr>
<tr>
<td>w1 &amp; w2</td>
<td>w3</td>
</tr>
<tr>
<td>?w1, w2</td>
<td>w3 &amp; w4</td>
</tr>
<tr>
<td>abc = def ghi &amp; jkl = mno</td>
<td>((abc = def ghi) &amp; (jkl=mno)</td>
</tr>
<tr>
<td>dog and cat WITHIN body</td>
<td>dog and (cat WITHIN body)</td>
</tr>
</tbody>
</table>

In the first example, because AND has a higher precedence than OR, the query returns all documents that contain w1 and all documents that contain both w2 and w3.

In the second example, the query returns all documents that contain both w1 and w2 and all documents that contain w3.
In the third example, the fuzzy operator is first applied to $w1$, then the AND operator is applied to arguments $w3$ and $w4$, then the OR operator is applied to term $w2$ and the results of the AND operation, and finally, the score from the fuzzy operation on $w1$ is added to the score from the OR operation.

The fourth example shows that the equivalence operator has higher precedence than the AND operator.

The fifth example shows that the AND operator has lower precedence than the WITHIN operator.

**Altering Precedence**

Precedence is altered by grouping characters as follows:

- Within parentheses, expansion or execution of operations is resolved before other expansions regardless of operator precedence.
- Within parentheses, precedence of operators is maintained during evaluation of expressions.
- Within parentheses, expansion operators are not applied to expressions unless the operators are also within the parentheses.

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

In all languages, an ABOUT query increases the number of relevant documents returned from the same query without this operator. Oracle Text scores results for an ABOUT query with the most relevant document receiving the highest score.

English and French Behavior

In English and French, use the ABOUT operator to query on concepts. The system looks up concept information in the theme component of the index. You create a theme component to your index by setting the INDEX_THEMES BASIC_LEXER attribute to YES.

Oracle Text retrieves documents that contain concepts that are related to your query word or phrase. For example, if you issue an ABOUT query on California, the system might return documents that contain the terms Los Angeles and San Francisco, which are cities in California. The document need not contain the term California to be returned in this ABOUT query.

The word or phrase specified in your ABOUT query need not exactly match the themes stored in the index. Oracle Text normalizes the word or phrase before performing lookup in the index.

You can use the ABOUT operator with the CONTAINS and CATSEARCH SQL operators. In the case of CATSEARCH, you must use query templating with the CONTEXT grammar to query on the indexed themes. See ABOUT Query with CATSEARCH in the Examples section.

---

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

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>about(phrase)</td>
<td>In all languages, increases the number of relevant documents returned for the same query without the ABOUT operator. The phrase parameter can be a single word or a phrase, or a string of words in free text format. In English and French, returns documents that contain concepts related to phrase, provided the BASIC_LEXER INDEX THEMES attribute is set to YES at index time. The score returned is a relevance score. Oracle Text ignores any query operators that are included in phrase. If your index contains only theme information, an ABOUT operator and operand must be included in your query on the text column or else Oracle Text returns an error. The phrase you specify cannot be more than 4000 characters.</td>
</tr>
</tbody>
</table>

Case-Sensitivity

ABOUT queries give the best results when your query is formulated with proper case. This is because the normalization of your query is based on the knowledge catalog which is case-sensitive.

However, you need not type your query in exact case to obtain results from an ABOUT query. The system does its best to interpret your query. For example, if you enter a query of CISCO and the system does not find this in the knowledge catalog, the system might use Cisco as a related concept for look-up.

Improving ABOUT Results

The ABOUT operator uses the supplied knowledge base in English and French to interpret the phrase you enter. Your ABOUT query therefore is limited to knowing and interpreting the concepts in the knowledge base.

You can improve the results of your ABOUT queries by adding your application-specific terminology to the knowledge base.

See Also: Extending the Knowledge Base in Chapter 14, "Executables".

3-6 Oracle Text Reference
Limitations

- The phrase you specify in an ABOUT query cannot be more than 4000 characters.

Examples

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

'about(soccer)'

Phrases
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 Text returns all documents that have themes of soccer, rules, or international competition.

In terms of scoring, documents which have all three themes will generally score higher than documents that have only one or two of the themes.

Unstructured Phrases
You can also query on unstructured phrases, such as the following:

'about(japanese banking investments in indonesia)'

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

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

'about(dogs) and cat'

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

'about(dogs) not about(labradors)'

Note: You cannot combine ABOUT with the WITHIN operator, as for example 'ABOUT (xyz) WITHIN abc'.

CONTAINS Query Operators 3-7
ABOUT Query with CATSEARCH
You can issue ABOUT queries with CATSEARCH using the query template method with grammar set to CONTEXT as follows:

```sql
select pk || ' ==> ' || text from test
where catsearch(text,
  '<query>
   <textquery grammar="context">
     about(California)
   </textquery>
   <score datatype="integer"/>
  </query>', '') > 0
order by pk;
```
ACCUMulate ( , )

Use the ACCUM operator to search for documents that contain at least one occurrence of any of the query terms. The accumulate operator ranks documents according to the total term weight of a document.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term1,term2</td>
<td>Returns documents that contain term1 or term2. Ranks documents</td>
</tr>
<tr>
<td>term1 accum term2</td>
<td>according to document term weight, with the highest scores assigned to</td>
</tr>
<tr>
<td></td>
<td>documents that have the highest total term weight.</td>
</tr>
</tbody>
</table>

Examples

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

'soccer, Brazil, cup'

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

'soccer, Brazil*3, cup'

Notes

Accumulate Scoring

ACCUM scores documents based on two criteria:

- document term weights
- document term scores

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

Accumulate scoring guarantees that if a document A matches $p$ terms with a total term weight of $m$, and document B matches $q$ terms with a total term weight of $m+1$, ...
document B is guaranteed to have a higher relevance score than document A, regardless of the numbers \( p \) and \( q \).

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

This following table illustrates accumulate scoring:

<table>
<thead>
<tr>
<th>Document</th>
<th>query</th>
<th>Score(x)</th>
<th>Score(y)</th>
<th>Score(z)</th>
<th>Total Term Weight</th>
<th>Score(query)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>x,y,z</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>x,y,z</td>
<td>10</td>
<td>20</td>
<td>0</td>
<td>2</td>
<td>38</td>
</tr>
<tr>
<td>C</td>
<td>x,y,z</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>3</td>
<td>73</td>
</tr>
<tr>
<td>D</td>
<td>x,y,z</td>
<td>50</td>
<td>50</td>
<td>0</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>E</td>
<td>x, y^*3, z</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>F</td>
<td>x, y^*3, z</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>41</td>
</tr>
</tbody>
</table>

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

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

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

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

Use the AND operator to search for documents that contain at least one occurrence of each of the query terms.

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>term1&amp;term2</code></td>
<td>Returns documents that contain <code>term1</code> and <code>term2</code>. Returns the minimum score of its operands. All query terms must occur; lower score taken.</td>
</tr>
<tr>
<td><code>term1 and term2</code></td>
<td></td>
</tr>
</tbody>
</table>

**Examples**

To obtain all the documents that contain the terms blue and black and red, issue the following query:

'blue & black & red'

In an AND query, the score returned is the score of the lowest query term. In this example, if the three individual scores for the terms blue, black, and red is 10, 20 and 30 within a document, the document scores 10.
Broader Term (BT, BTG, BTP, BTI)

Use the broader term operators (BT, BTG, BTP, BTI) to expand a query to include the term that has been defined in a thesaurus as the broader or higher level term for a specified term. They can also expand the query to include the broader term for the broader term and the broader term for that broader term, and so on up through the thesaurus hierarchy.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT(term)((qualifier))</td>
<td>Expands a query to include the term defined in the thesaurus as a broader term for term.</td>
</tr>
<tr>
<td>BTG(term)((qualifier))</td>
<td>Expands a query to include all terms defined in the thesaurus as broader generic terms for term.</td>
</tr>
<tr>
<td>BTP(term)((qualifier))</td>
<td>Expands a query to include all the terms defined in the thesaurus as broader partitive terms for term.</td>
</tr>
<tr>
<td>BTI(term)((qualifier))</td>
<td>Expands a query to include all the terms defined in the thesaurus as broader instance terms for term.</td>
</tr>
</tbody>
</table>

**term**

Specify the operand for the broader term operator. Oracle Text expands term to include the broader term entries defined for the term in the thesaurus specified by thes. For example, if you specify BTG(dog), the expansion includes only those terms that are defined as broader term generic for dog. You cannot specify expansion operators in the term argument.

The number of broader terms included in the expansion is determined by the value for level.

**qualifier**

Specify a qualifier for term, if term is a homograph (word or phrase with multiple meanings, but the same spelling) that appears in two or more nodes in the same hierarchy branch of thes.

If a qualifier is not specified for a homograph in a broader term query, the query expands to include the broader terms of all the homographic terms.
level
Specify the number of levels traversed in the thesaurus hierarchy to return the broader terms for the specified term. For example, a level of 1 in a BT query returns the broader term entry, if one exists, for the specified term. A level of 2 returns the broader term entry for the specified term, as well as the broader term entry, if one exists, for the broader term.

The level argument is optional and has a default value of one (1). Zero or negative values for the level argument return only the original query term.

thes
Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. A thesaurus named DEFAULT must exist in the thesaurus tables if you use this default value.

---

**Note:** If you specify thes, you must also specify level.

---

Examples

The following query returns all documents that contain the term tutorial or the BT term defined for tutorial in the DEFAULT thesaurus:

'BT(tutorial)'

When you specify a thesaurus name, you must also specify level as in:

'BT(tutorial, 2, mythes)'

**Broader Term Operator on Homographs**

If machine is a broader term for crane (building equipment) and bird is a broader term for crane (waterfowl) and no qualifier is specified for a broader term query, the query

BT(crane)

expands to:

'({crane} or {machine} or {bird})'

If waterfowl is specified as a qualifier for crane in a broader term query, the query

BT(crane{(waterfowl)})

expands to the query:
Broader Term (BT, BTG, BTP, BTI)

'(crane) or (bird)'

**Note:** When specifying a qualifier in a broader or narrower term query, the qualifier and its notation (parentheses) must be escaped, as is shown in this example.

**Related Topics**

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

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

Use the EQUIV operator to specify an acceptable substitution for a word in a query.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term1=term2</td>
<td>Specifies that term2 is an acceptable substitution for term1. Score calculated as the sum of all occurrences of both terms.</td>
</tr>
<tr>
<td>term1 equiv term2</td>
<td></td>
</tr>
</tbody>
</table>

Examples

The following example returns all documents that contain either the phrase alsatians are big dogs or labradors are big dogs:

'labradors=alsatians are big dogs'

Operator Precedence

The EQUIV operator has higher precedence than all other operators except the expansion operators (fuzzy, soundex, stem).
Fuzzy

Use the fuzzy operator to expand queries to include words that are spelled similarly to the specified term. This type of expansion is helpful for finding more accurate results when there are frequent misspellings in your document set.

The new fuzzy syntax enables you to rank the result set so that documents that contain words with high similarity to the query word are scored higher than documents with lower similarity. You can also limit the number of expanded terms.

Unlike stem expansion, the number of words generated by a fuzzy expansion depends on what is in the index. Results can vary significantly according to the contents of the index.

Supported Languages

Oracle Text supports fuzzy definitions for English, German, Italian, Dutch, Spanish, Japanese, and OCR.

Stopwords

If the fuzzy expansion returns a stopword, the stopword is not included in the query or highlighted by `CTX_DOC.HIGHLIGHT` or `CTX_DOC.MARKUP`.

Base-Letter Conversion

If base-letter conversion is enabled for a text column and the query expression contains a fuzzy operator, Oracle Text operates on the base-letter form of the query.

Syntax

```
fuzzy(term, score, numresults, weight)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term</td>
<td>Specify the word on which to perform the fuzzy expansion. Oracle Text expands <code>term</code> to include words only in the index.</td>
</tr>
<tr>
<td>score</td>
<td>Specify a similarity score. Terms in the expansion that score below this number are discarded. Use a number between 1 and 80. The default is 60.</td>
</tr>
<tr>
<td>numresults</td>
<td>Specify the maximum number of terms to use in the expansion of <code>term</code>. Use a number between 1 and 5000. The default is 100.</td>
</tr>
</tbody>
</table>
Consider the CONTAINS query:

```sql
...CONTAINS(TEXT, 'fuzzy(government, 70, 6, weight)', 1) > 0;
```

This query expands to the first six fuzzy variations of `government` in the index that have a similarity score over 70.

In addition, documents in the result set are weighted according to their similarity to `government`. Documents containing words most similar to government receive the highest score.

You can skip unnecessary parameters using the appropriate number of commas. For example:

'fuzzy(government,,,weight)'

### Backward Compatibility Syntax

The old fuzzy syntax from previous releases is still supported. This syntax is as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?term</td>
<td>Expands term to include all terms with similar spellings as the specified term.</td>
</tr>
</tbody>
</table>
HASPATH

Use this operator to find all XML documents that contain a specified section path. You can also use this operator to do section equality testing.

Your index must be created with the PATH_SECTION_GROUP for this operator to work.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASPATH(path)</td>
<td>Searches an XML document set and returns a score of 100 for all documents where path exists. Separate parent and child paths with the / character. For example, you can specify A/B/C. See example.</td>
</tr>
<tr>
<td>HASPATH(A=&quot;value&quot;)</td>
<td>Searches an XML document set and returns a score of 100 for all documents that have the element A with content value and only value. See example.</td>
</tr>
</tbody>
</table>

Example

**Path Testing**
The query

```plaintext
HASPATH(A/B/C)
```

finds and returns a score of 100 for the document

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

without the query having to reference dog at all.

**Section Equality Testing**
The query

```plaintext
dog INPATH A
```

finds
<h3>HASPATH</h3>

\(<\text{A}>\text{dog}</\text{A}>\)

but it also finds
\(<\text{A}>\text{dog park}</\text{A}>\)

To limit the query to the term \textit{dog} and nothing else, you can use a section equality test with the \texttt{HASPATH} operator. For example,

\begin{verbatim}
HASPATH (A="dog")
\end{verbatim}

finds and returns a score of 100 only for the first document, and not the second.

**Limitations**

Because of how XML section data is recorded, false matches might occur with XML sections that are completely empty as follows:

\(<\text{A}><\text{B}></\text{C}></\text{B}></\text{D}><\text{E}></\text{E}></\text{D}></\text{A}>\)

A query of \texttt{HASPATH (A/B/E)} or \texttt{HASPATH (A/D/C)} falsely matches this document. This type of false matching can be avoided by inserting text between empty tags.

The typical document could be:

\(<\text{A}>\text{dog}</\text{A}>\)

but it also finds
\(<\text{A}>\text{dog park}</\text{A}>\)

To limit the query to the term \textit{dog} and nothing else, you can use a section equality test with the \texttt{HASPATH} operator. For example,

\begin{verbatim}
HASPATH (A="dog")
\end{verbatim}

finds and returns a score of 100 only for the first document, and not the second.

**Limitations**

Because of how XML section data is recorded, false matches might occur with XML sections that are completely empty as follows:

\(<\text{A}><\text{B}></\text{C}></\text{B}></\text{D}><\text{E}></\text{E}></\text{D}></\text{A}>\)

A query of \texttt{HASPATH (A/B/E)} or \texttt{HASPATH (A/D/C)} falsely matches this document. This type of false matching can be avoided by inserting text between empty tags.
INPATH

Use this operator to do path searching in XML documents. This operator is like the WITHIN operator except that the right-hand side is a parentheses enclosed path, rather than a single section name.

Your index must be created with the PATH_SECTION_GROUP for the INPATH operator to work.

Syntax

The INPATH operator has the following syntax:

Top-Level Tag Searching

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term INPATH (/A)</td>
<td>Returns documents that have <code>term</code> within the <code>&lt;A&gt;</code> and <code>&lt;/A&gt;</code> tags.</td>
</tr>
<tr>
<td>term INPATH (A)</td>
<td></td>
</tr>
</tbody>
</table>

Any-Level Tag Searching

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term INPATH (//A)</td>
<td>Returns documents that have <code>term</code> in the <code>&lt;A&gt;</code> tag at any level. This query is the same as <code>term WITHIN A'</code>.</td>
</tr>
</tbody>
</table>

Direct Parentage Path Searching

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term INPATH (A/B)</td>
<td>Returns documents where <code>term</code> appears in a B element which is a direct child of a top-level A element. For example, a document containing <code>&lt;A&gt;&lt;B&gt;term&lt;/B&gt;&lt;/A&gt;</code> is returned.</td>
</tr>
</tbody>
</table>
## Single-Level Wildcard Searching

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>term INPATH (A/*/B)</code></td>
<td>Returns documents where <em>term</em> appears in a B element which is a grandchild (two levels down) of a top-level A element. For example, a document containing <code>&lt;A&gt;&lt;D&gt;&lt;B&gt;term&lt;/B&gt;&lt;/D&gt;&lt;/A&gt;</code> is returned.</td>
</tr>
</tbody>
</table>

## Multi-level Wildcard Searching

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>term INPATH (A/*/B/*//*/C)</code></td>
<td>Returns documents where <em>term</em> appears in a C element which is 3 levels down from a B element which is two levels down (grandchild) of a top-level A element.</td>
</tr>
</tbody>
</table>

## Any-Level Descendant Searching

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>term INPATH (A//B)</code></td>
<td>Returns documents where <em>term</em> appears in a B element which is some descendant (any level) of a top-level A element.</td>
</tr>
</tbody>
</table>

## Attribute Searching

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>term INPATH (/A/@B)</code></td>
<td>Returns documents where <em>term</em> appears in the B attribute of an A element at any level. Attributes must be bound to a direct parent.</td>
</tr>
</tbody>
</table>
Descendant/Attribute Existence Testing

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term INPATH (A[B])</td>
<td>Returns documents where term appears in a top-level A element which has a B element as a direct child.</td>
</tr>
<tr>
<td>term INPATH (A[./B])</td>
<td>Returns documents where term appears in a top-level A element which has a B element as a descendant at any level.</td>
</tr>
<tr>
<td>term INPATH (/A[@B])</td>
<td>Finds documents where term appears in an A element at any level which has a B attribute. Attributes must be tied to a direct parent.</td>
</tr>
</tbody>
</table>

Attribute Value Testing

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term INPATH (A[@B = &quot;value&quot;]))</td>
<td>Finds all documents where term appears in a top-level A element which has a B attribute whose value is value.</td>
</tr>
<tr>
<td>term INPATH (A[@B != &quot;value&quot;]))</td>
<td>Finds all documents where term appears in a top-level A element which has a B attribute whose value is not value.</td>
</tr>
</tbody>
</table>

Tag Value Testing

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term INPATH (A[B = &quot;value&quot;]))</td>
<td>Returns documents where term appears in an A tag which has a B tag whose value is value.</td>
</tr>
</tbody>
</table>

Not

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term INPATH (A[NOT(B)])</td>
<td>Finds documents where term appears in a top-level A element which does not have a B element as an immediate child.</td>
</tr>
</tbody>
</table>
AND and OR Testing

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term INPATH (A[B and C])</td>
<td>Finds documents where term appears in a top-level A element which has a B and a C element as an immediate child.</td>
</tr>
<tr>
<td>term INPATH (A[B and @C=&quot;value&quot;]])</td>
<td>Finds documents where <em>term</em> appears in a top-level A element which has a B element and a C attribute whose value is <em>value</em>.</td>
</tr>
<tr>
<td>term INPATH (A [B OR C])</td>
<td>Finds documents where <em>term</em> appears in a top-level A element which has a B element or a C element.</td>
</tr>
</tbody>
</table>

Combining Path and Node Tests

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term INPATH (A[@B = &quot;value&quot;]/C/D)</td>
<td>Returns documents where <em>term</em> appears in a D element which is the child of a C element, which is the child of a top-level A element with a B attribute whose value is <em>value</em>.</td>
</tr>
</tbody>
</table>

Nested INPATH

You can nest the entire INPATH expression in another INPATH expression as follows:

\[(\text{dog INPATH (//A/B/C)) INPATH (D)}\]

When you do so, the two INPATH paths are completely independent. The outer INPATH path does not change the context node of the inner INPATH path. For example:

\[(\text{dog INPATH (A)) INPATH (D)}\]

never finds any documents, because the inner INPATH is looking for *dog* within the top-level tag A, and the outer INPATH constrains that to document with top-level tag D. A document can have only one top-level tag, so this expression never finds any documents.

Case-Sensitivity

Tags and attribute names in path searching are case-sensitive. That is,
Examples

**Top-Level Tag Searching**
To find all documents that contain the term *dog* in the top-level tag `<A>`:

```sql
dog INPATH (/A)
or
dog INPATH(A)
```

This query finds the following documents:

```xml
<A>dog</A>
```

and

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

**Any-Level Tag Searching**
To find all documents that contain the term *dog* in the `<A>` tag at any level:

```sql
dog INPATH(//A)
```

This query finds the following documents:

```xml
<A>dog</A>
```

**Direct Parentage Searching**
To find all documents that contain the term *dog* in a B element that is a direct child of a top-level A element:

```sql
dog INPATH(A/B)
```

This query finds the following XML document:

```xml
<A><B>My dog is friendly.</B><A>
```

but does not find:

```xml
<C><B>My dog is friendly.</B></C>
```

**Tag Value Testing**
You can test the value of tags. For example, the query:

```sql
dog INPATH (A)
```
dog INPATH(A[B="dog"])

Finds the following document:
<A><B>dog</B></A>

But does not find:
<A><B>My dog is friendly.</B></A>

**Attribute Searching**
You can search the content of attributes. For example, the query:
dog INPATH(//A/@B)

Finds the document
<C><A B="snoop dog"> </A> </C>

**Attribute Value Testing**
You can test the value of attributes. For example, the query
California INPATH (//A[@B = "home address"])  

Finds the document:
<A B="home address">San Francisco, California, USA</A>
But does not find:
<A B="work address">San Francisco, California, USA</A>

**Path Testing**
You can test if a path exists with the HASPATH operator. For example, the query:
HASPATH(A/B/C)
finds and returns a score of 100 for the document
<A><B><C>dog</C></B></A>
without the query having to reference dog at all.
Limitations

Testing for Equality
The following is an example of an INPATH equality test.

dog INPATH (@B = "foo")

The following limitations apply for these expressions:

- Only equality and inequality are supported. Range operators and functions are not supported.
- The left hand side of the equality must be an attribute. Tags and literals here are not enabled.
- The right hand side of the equality must be a literal. Tags and attributes here are not allowed.
- The test for equality depends on your lexer settings. With the default settings, the query
  
dog INPATH (@B = "pot of gold")

matches the following sections:

- `<A B="POT OF GOLD">dog</A>`
- `<A B="pot of gold">dog</A>` because lexer is case-insensitive by default.
- `<A B="POT IS GOLD">dog</A>` because `of` and `is` are default stopwords in English, and a stopword matches any stopword word.
- `<A B="POT_OF_GOLD">dog</A>` because the underscore character is not a join character by default.
Use the MDATA operator to query documents that contain MDATA sections. MDATA sections are metadata that have been added to documents to speed up mixed querying.

MDATA queries are treated exactly as literals. For example, with the query

\[
\text{MDATA(price, } \$1.24)\]

the \$ is not interpreted as a stem operator, nor is the . (period) transformed into whitespace. A right (close) parenthesis terminates the MDATA operator, so that MDATA values that have close parentheses cannot be searched.

**Syntax**

**Syntax**

\[
\text{MDATA(sectionname, value)}
\]

*sectionname*

The name of the MDATA section(s) to search.

*value*

The value of the MDATA section. For example, if an MDATA section called *Booktype* has been created, it might have a value of *paperback*.

**Example**

Suppose you want to query for books written by the writer Nigella Lawson that contain the word *summer*. Assuming that an MDATA section called *AUTHOR* has been declared, you can query as follows:

```sql
SELECT id FROM idx_docs
WHERE CONTAINS(text, 'summer AND MDATA(author, Nigella Lawson)')>0
```

This query will only be successful if an AUTHOR tag has the exact value *Nigella Lawson* (after simplified tokenization). *Nigella* or *Ms. Nigella Lawson* will not work.
**Notes**

MDATA query values ignore stopwords.

The **MDATA** operator returns 100 or 0, depending on whether the document is a match.

The **MDATA** operator is not supported for **CTXCAT**, **CTXRULE**, or **CTXXMLPATH** indexes.

Table 3–1 shows how MDATA interacts with some other query operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Allowed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>dog &amp; MDATA(a, b)</td>
<td>yes</td>
</tr>
<tr>
<td>OR</td>
<td>dog</td>
<td>MDATA(a, b)</td>
</tr>
<tr>
<td>NOT</td>
<td>dog ~ MDATA(a, b)</td>
<td>yes</td>
</tr>
<tr>
<td>MINUS</td>
<td>dog - MDATA(a, b)</td>
<td>yes</td>
</tr>
<tr>
<td>ACCUM</td>
<td>dog , MDATA(a, b)</td>
<td>yes</td>
</tr>
<tr>
<td>PHRASE</td>
<td>MDATA(a, b) dog</td>
<td>no</td>
</tr>
<tr>
<td>NEAR</td>
<td>MDATA(a, b) ; dog</td>
<td>no</td>
</tr>
<tr>
<td>WITHIN, HAPATH, INPATH</td>
<td>MDATA(a, b) WITHIN c</td>
<td>no</td>
</tr>
</tbody>
</table>

Thesaurus expansion

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Allowed?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MDATA(a, SYN(b))</td>
<td>no (syntactically allowed, but the inner operator is treated as literal text)</td>
</tr>
<tr>
<td></td>
<td>MDATA(a, $b)</td>
<td>no (syntactically allowed, but the inner operator is treated as literal text)</td>
</tr>
<tr>
<td></td>
<td>MDATA(a, b%)</td>
<td>no (syntactically allowed, but the inner operator is treated as literal text)</td>
</tr>
<tr>
<td></td>
<td>MDATA(a, ![b])</td>
<td>no (syntactically allowed, but the inner operator is treated as literal text)</td>
</tr>
<tr>
<td></td>
<td>MDATA(a, ?b)</td>
<td>no (syntactically allowed, but the inner operator is treated as literal text)</td>
</tr>
<tr>
<td>ABOUT</td>
<td>ABOUT(MDATA(a,b))</td>
<td>no (syntactically allowed, but the inner operator is treated as literal text)</td>
</tr>
<tr>
<td></td>
<td>MDATA(ABOUT(a))</td>
<td>no (syntactically allowed, but the inner operator is treated as literal text)</td>
</tr>
</tbody>
</table>

When MDATA sections repeat, each instance is a separate and independent value.

For instance, the document

<AUTHOR>Terry Pratchett</AUTHOR><AUTHOR>Douglas Adams</AUTHOR>

can be found with any of the following queries:
MDATA(author, Terry Pratchett)
MDATA(author, Douglas Adams)
MDATA(author, Terry Pratchett) and MDATA(author, Douglas Adams)

but not any of the following:

MDATA(author, Terry Pratchett Douglas Adams)
MDATA(author, Terry Pratchett & Douglas Adams)
MDATA(author, Pratchett Douglas)

Related Topics

See also "ADD_MDATA" on page 7-11 and "ADD_MDATA_SECTION" on page 7-14, as well as the Section Searching chapter of the Oracle Text Application Developer’s Guide.
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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term1-term2</td>
<td>Returns documents that contain term1. Calculates score by subtracting the score of term2 from the score of term1. Only documents with positive score are returned.</td>
</tr>
<tr>
<td>term1 minus term2</td>
<td></td>
</tr>
</tbody>
</table>

Examples

Suppose a query on the term cars always returned high scoring documents about Ford cars. You can lower the scoring of the Ford documents by using the expression:

'cars - Ford'

In essence, this expression returns documents that contain the term cars and possibly Ford. However, the score for a returned document is the score of cars minus the score of Ford.
Narrower Term (NT, NTG, NTP, NTI)

Use the narrower term operators (NT, NTG, NTP, NTI) to expand a query to include all the terms that have been defined in a thesaurus as the narrower or lower level terms for a specified term. They can also expand the query to include all of the narrower terms for each narrower term, and so on down through the thesaurus hierarchy.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT(term[(qualifier)][,level][,thes])</td>
<td>Expands a query to include all the lower level terms defined in the thesaurus as narrower terms for term.</td>
</tr>
<tr>
<td>NTG(term[(qualifier)][,level][,thes])</td>
<td>Expands a query to include all the lower level terms defined in the thesaurus as narrower generic terms for term.</td>
</tr>
<tr>
<td>NTP(term[(qualifier)][,level][,thes])</td>
<td>Expands a query to include all the lower level terms defined in the thesaurus as narrower partitive terms for term.</td>
</tr>
<tr>
<td>NTI(term[(qualifier)][,level][,thes])</td>
<td>Expands a query to include all the lower level terms defined in the thesaurus as narrower instance terms for term.</td>
</tr>
</tbody>
</table>

**term**

Specify the operand for the narrower term operator. term is expanded to include the narrower term entries defined for the term in the thesaurus specified by thes. The number of narrower terms included in the expansion is determined by the value for level. You cannot specify expansion operators in the term argument.

**qualifier**

Specify a qualifier for term, if term is a homograph (word or phrase with multiple meanings, but the same spelling) that appears in two or more nodes in the same hierarchy branch of thes.

If a qualifier is not specified for a homograph in a narrower term query, the query expands to include all of the narrower terms of all homographic terms.
level
Specify the number of levels traversed in the thesaurus hierarchy to return the narrower terms for the specified term. For example, a level of 1 in an NT query returns all the narrower term entries, if any exist, for the specified term. A level of 2 returns all the narrower term entries for the specified term, as well as all the narrower term entries, if any exist, for each narrower term.

The level argument is optional and has a default value of one (1). Zero or negative values for the level argument return only the original query term.

thes
Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. A thesaurus named DEFAULT must exist in the thesaurus tables if you use this default value.

Note: If you specify thes, you must also specify level.

Examples

The following query returns all documents that contain either the term cat or any of the NT terms defined for cat in the DEFAULT thesaurus:

'NT(cat)'

If you specify a thesaurus name, you must also specify level as in:

'NT(cat, 2, mythes)'

The following query returns all documents that contain either fairy tale or any of the narrower instance terms for fairy tale as defined in the DEFAULT thesaurus:

'NTI(fairy tale)'

That is, if the terms cinderella and snow white are defined as narrower term instances for fairy tale, Oracle Text returns documents that contain fairy tale, cinderella, or snow white.

Notes

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

**Related Topics**

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

**See Also:** For more information on browsing the narrower terms in your thesaurus, see CTX_Thes.NT in Chapter 12, "CTX_Thes Package".
Use the `NEAR` operator to return a score based on the proximity of two or more query terms. Oracle Text returns higher scores for terms closer together and lower scores for terms farther apart in a document.

**Note:** The `NEAR` operator works with only word queries. You cannot use `NEAR` in `ABOUT` queries.

### Syntax

```sql
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 and may make use of other query operators (see "NEAR with Other Operators").

**max_span**

Optionally specify the size of the biggest clump. The default is 100. Oracle Text returns an error if you specify a number greater than 100.

A clump is the smallest group of words in which all query terms occur. All clumps begin and end with a query term.

For near queries with two terms, `max_span` is the maximum distance allowed between the two terms. For example, to query on `dog` and `cat` where `dog` is within 6 words of `cat`, issue the following query:

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

**order**

Specify `TRUE` for Oracle Text to search for terms in the order you specify. The default is `FALSE`.

For example, to search for the words `monday`, `tuesday`, and `wednesday` in that order with a maximum clump size of 20, issue the following query:

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

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

Oracle Text might return different scores for the same document when you use identical query expressions that have the order flag set differently. For example, Oracle Text might return different scores for the same document when you 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 Text returns a score between 1 and 100 that is proportional to the number of clumps in the document and inversely proportional to the average size of the clumps. This means many small clumps in a document result in higher scores, 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.

The size of a clump does not include the query terms themselves. So for the query NEAR((DOG, CAT), 1), dog cat will be a match, and dog ate cat will be a match, but dog sat on cat will not be a match.

NEAR with Other Operators

You can use the NEAR operator with other operators such as AND and OR. Scores are calculated in the regular way.

For example, to find all documents that contain the terms tiger, lion, and cheetah where the terms lion and tiger are within 10 words of each other, 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*.

The following operators also work with NEAR:

- EQUIV
- NEAR itself
- All expansion operators that produce words, phrases, or EQUIV. These include:
  - soundex
  - fuzzy
  - wildcards
  - stem

**Backward Compatibility NEAR Syntax**

You can write near queries using the syntax of previous Oracle Text releases. For example, to find all documents where *lion* occurs near *tiger*, you can write:

\`'lion near tiger'\`

or with the semi-colon as follows:

\`'lion;tiger'\`

This query is equivalent to the following query:

\`'near((lion, tiger), 100, FALSE)'\`

**Note:** Only the syntax of the NEAR operator is backward compatible. In the example, the score returned is calculated using the clump method as described in this section.
Highlighting with the NEAR Operator

When you use highlighting and your query contains the near operator, all occurrences of all terms in the query that satisfy the proximity requirements are highlighted. Highlighted terms can be single words or phrases.

For example, assume a document contains the following text:

Chocolate and vanilla are my favorite ice cream flavors. I like chocolate served in a waffle cone, and vanilla served in a cup with carmel syrup.

If the query is `near((chocolate, vanilla), 100, FALSE)`, the following is highlighted:

<<Chocolate>> and <<vanilla>> are my favorite ice cream flavors. I like chocolate served in a waffle cone, and vanilla served in a cup with carmel syrup.

However, if the query is `near((chocolate, vanilla), 4, FALSE)`, only the following is highlighted:

<<Chocolate>> and <<vanilla>> are my favorite ice cream flavors. I like chocolate served in a waffle cone, and vanilla served in a cup with carmel syrup.

See Also: For more information about the procedures you can use for highlighting, see Chapter 8, "CTX_DOC Package".

Section Searching and NEAR

You can use the NEAR operator with the WITHIN operator for section searching as follows:

'near((dog, cat), 10) WITHIN Headings'

When evaluating expressions such as these, Oracle Text looks for clumps that lie entirely within the given section.

In this example, only those clumps that contain dog and cat that lie entirely within the section Headings are counted. That is, if the term dog lies within Headings and the term cat lies five words from dog, but outside of Headings, this pair of words does not satisfy the expression and is not counted.
NOT (~)

Use the NOT operator to search for documents that contain one query term and not another.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>term1~term2</code></td>
<td>Returns documents that contain <code>term1</code> and not <code>term2</code>.</td>
</tr>
<tr>
<td><code>term1 not term2</code></td>
<td></td>
</tr>
</tbody>
</table>

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.
Use the OR operator to search for documents that contain at least one occurrence of any of the query terms.

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term1</td>
<td>term2</td>
</tr>
<tr>
<td>term1 or term2</td>
<td></td>
</tr>
</tbody>
</table>

**Examples**

For example, to obtain the documents that contain the term cats or the term dogs, use either of the following expressions:

'cats | dogs'
'cats OR dogs'

**Scoring**

In an OR query, the score returned is the score for the highest query term. In the example, if the scores for cats and dogs is 30 and 40 within a document, the document scores 40.
Preferred Term (PT)

Use the preferred term operator (PT) to replace a term in a query with the preferred term that has been defined in a thesaurus for the term.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT(term[,thes])</td>
<td>Replaces the specified word in a query with the preferred term for term.</td>
</tr>
</tbody>
</table>

**term**

Specify the operand for the preferred term operator. term is replaced by the preferred term defined for the term in the specified thesaurus. However, if no PT entries are defined for the term, term is not replaced in the query expression and term is the result of the expansion.

You cannot specify expansion operators in the term argument.

**thes**

Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. As a result, a thesaurus named DEFAULT must exist in the thesaurus tables before using any of the thesaurus operators.

Examples

The term automobile has a preferred term of car in a thesaurus. A PT query for automobile returns all documents that contain the word car. Documents that contain the word automobile are not returned.

Related Topics

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

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

Use the related term operator (RT) to expand a query to include all related terms that have been defined in a thesaurus for the term.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT(term[,thes])</td>
<td>Expands a query to include all the terms defined in the thesaurus as a related term for term.</td>
</tr>
</tbody>
</table>

**term**

Specify the operand for the related term operator. term is expanded to include term and all the related entries defined for term in thes.

You cannot specify expansion operators in the term argument.

**thes**

Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. As a result, a thesaurus named DEFAULT must exist in the thesaurus tables before using any of the thesaurus operators.

Examples

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

Related Topics

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

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

Use the soundex (!) operator to expand queries to include words that have similar sounds; that is, words that sound like other words. This function enables comparison of words that are spelled differently, but sound alike in English.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!term</td>
<td>Expands a query to include all terms that sound the same as the specified term (English-language text only).</td>
</tr>
</tbody>
</table>

Examples

```sql
SELECT ID, COMMENT FROM EMP_RESUME
WHERE CONTAINS (COMMENT, '!SMYTHE') > 0 ;
```

<table>
<thead>
<tr>
<th>ID</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Smith is a hard worker who..</td>
</tr>
</tbody>
</table>

Language

Soundex works best for languages that use a 7-bit character set, such as English. It can be used, with lesser effectiveness, for languages that use an 8-bit character set, such as many Western European languages.

If you have base-letter conversion specified for a text column and the query expression contains a soundex operator, Oracle Text operates on the base-letter form of the query.
stem ($) 

Use the stem ($) operator to search for terms that have the same linguistic root as the query term.

If you use the BASIC_LEXER to index your language, stemming performance can be improved by using the index_stems attribute.

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

Japanese stemming is supported with the JAPANESE_LEXER.

You can specify your stemming language with the BASIC_WORDLIST wordlist preference.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$term</td>
<td>Expands a query to include all terms having the same stem or root word as the specified term.</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>Input</th>
<th>Expands To</th>
</tr>
</thead>
<tbody>
<tr>
<td>$scream</td>
<td>scream screaming screamed</td>
</tr>
<tr>
<td>$distinguish</td>
<td>distinguish distinguished distinguishes</td>
</tr>
<tr>
<td>$guitars</td>
<td>guitars guitar</td>
</tr>
<tr>
<td>$commit</td>
<td>commit committed</td>
</tr>
<tr>
<td>$cat</td>
<td>cat cats</td>
</tr>
<tr>
<td>$sing</td>
<td>sang sung sing</td>
</tr>
</tbody>
</table>

Behavior with Stopwords

If stem returns a word designated as a stopword, the stopword is not included in the query or highlighted by CTX_QUERY.HIGHLIGHT or CTX_QUERY.MARKUP.
Related Topics

**See Also:** For more information about enabling the stem operator with BASIC_LEXER, see BASIC_LEXER in Chapter 2, "Oracle Text Indexing Elements".
Stored Query Expression (SQE)

Use the SQE operator to call a stored query expression created with the \texttt{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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{SQE(SQE_name)}</td>
<td>Returns the results for the stored query expression \texttt{SQE_name}.</td>
</tr>
</tbody>
</table>

### Examples

To create an SQE named \textit{disasters}, use \texttt{CTX\_QUERY.\_STORE\_SQE} as follows:

\begin{verbatim}
begin
  ctx_query.store_sqe('disasters', 'hurricane or earthquake or blizzard');
end;
\end{verbatim}

This stored query expression returns all documents that contain either \textit{hurricane}, \textit{earthquake} or \textit{blizzard}.

\textit{This SQE} can then be called within a query expression as follows:

\begin{verbatim}
SELECT SCORE(1), docid FROM news
WHERE CONTAINS(resume, 's QE(disasters)', 1)> 0
ORDER BY SCORE(1);
\end{verbatim}
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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN(term[,thes])</td>
<td>Expands a query to include all the terms defined in the thesaurus as synonyms for term.</td>
</tr>
</tbody>
</table>

**term**
Specify the operand for the synonym operator. term is expanded to include term and all the synonyms defined for term in thes.

You cannot specify expansion operators in the term argument.

**thes**
Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. A thesaurus named DEFAULT must exist in the thesaurus tables if you use this default value.

Examples

The following query expression returns all documents that contain the term dog or any of the synonyms defined for dog in the DEFAULT thesaurus:

`'SYN(dog)'`

**Compound Phrases in Synonym Operator**

Expansion of compound phrases for a term in a synonym query are returned as AND conjunctives.

For example, the compound phrase temperature + measurement + instruments is defined in a thesaurus as a synonym for the term thermometer. In a synonym query for thermometer, the query is expanded to:

`(thermometer) OR ((temperature)&(measurement)&(instruments))`
Related Topics

You can browse your thesaurus using procedures in the CTX THES package.

See Also: For more information on browsing the synonym terms in your thesaurus, see CTX THES SYN in Chapter 12, "CTX THES Package".
Threshold (>)

Use the threshold operator (>) in two ways:

- at the expression level
- at the query term level

The threshold operator at the expression level eliminates documents in the result set that score below a threshold number.

The threshold operator at the query term level selects a document based on how a term scores in the document.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression&gt;n</td>
<td>Returns only those documents in the result set that score above the threshold n.</td>
</tr>
<tr>
<td>term&gt;n</td>
<td>Within an expression, returns documents that contain the query term with score of at least n.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR(term[, lang[, thes]])</td>
<td>Expands term to include all the foreign equivalents that are defined for term.</td>
</tr>
</tbody>
</table>

**term**

Specify the operand for the translation term operator. term is expanded to include all the foreign language entries defined for term in thes. You cannot specify expansion operators in the term argument.

**lang**

Optionally, specify which foreign language equivalents to return in the expansion. The language you specify must match the language as defined in thes. (You may specify only one language at a time.) If you omit this parameter or specify it as ALL, the system expands to use all defined foreign language terms.

**thes**

Optionally, specify the name of the thesaurus used to return the expansions for the specified term. The thes argument has a default value of DEFAULT. As a result, a thesaurus named DEFAULT must exist in the thesaurus tables before you can use any of the thesaurus operators.

**Note:** If you specify thes, you must also specify lang.

Examples

Consider a thesaurus MY_THES with the following entries for cat:

```
cat
   SPANISH: gato
   FRENCH: chat
```
To search for all documents that contain *cat* and the Spanish translation of *cat*, issue the following query:

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

This query expands to:

`'(cat)|{gato}'`

**Related Topics**

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

**See Also:** For more information on browsing the related terms in your thesaurus, see CTX_THES.TR in Chapter 12, "CTX_THES Package".
Translation Term Synonym (TRSYN)

Use the translation term operator (TR) to expand a query to include all the defined foreign equivalents of the query term, the synonyms of query term, and the foreign equivalents of the synonyms.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRSYN(term[, lang[, thes]])</td>
<td>Expands term to include foreign equivalents of term, the synonyms of term, and the foreign equivalents of the synonyms.</td>
</tr>
</tbody>
</table>

**term**
Specify the operand for this operator. term is expanded to include all the foreign language entries and synonyms defined for term in thes. You cannot specify expansion operators in the term argument.

**lang**
Optionally, specify which foreign language equivalents to return in the expansion. The language you specify must match the language as defined in thes. If you omit this parameter, the system expands to use all defined foreign language terms.

**thes**
Optionally, specify the name of the thesaurus used to return the expansions for the specified term. The thes argument has a default value of DEFAULT. As a result, a thesaurus named DEFAULT must exist in the thesaurus tables before you can use any of the thesaurus operators.

Note: If you specify thes, you must also specify lang.

Examples

Consider a thesaurus MY_THES with the following entries for cat:

```plaintext
cat
  _SPANISH: gato
  FRENCH: chat
  SYN lion
```
SPANISH: leon

To search for all documents that contain \textit{cat}, the Spanish equivalent of \textit{cat}, the synonym of \textit{cat}, and the Spanish equivalent of \textit{lion}, issue the following query:

\texttt{\textasciitilde trsyn(cat, \textit{spanish}, my\_thes)}

This query expands to:

\texttt{\{(cat)\|gato\|lion\|leon\}}

**Related Topics**

You can browse a thesaurus using procedures in the \texttt{CTX\_THES} package.

**See Also:** For more information on browsing the translation and synonym terms in your thesaurus, see \texttt{CTX\_THES\_TRSYN} in Chapter 12, "CTX\_THES Package".
Top Term (TT)

Use the top term operator (TT) to replace a term in a query with the top term that has been defined for the term in the standard hierarchy (BT, NT) in a thesaurus. Top terms in the generic (BTG, NTG), partitive (BTP, NTP), and instance (BTI, NTI) hierarchies are not returned.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT(term[,thes])</td>
<td>Replaces the specified word in a query with the top term in the standard hierarchy (BT, NT) for term.</td>
</tr>
</tbody>
</table>

**term**

Specify the operand for the top term operator. term is replaced by the top term defined for the term in the specified thesaurus. However, if no TT entries are defined for term, term is not replaced in the query expression and term is the result of the expansion.

You cannot specify expansion operators in the term argument.

**thes**

Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. A thesaurus named DEFAULT must exist in the thesaurus tables if you use this default value.

Examples

The term dog has a top term of animal in the standard hierarchy of a thesaurus. A TT query for dog returns all documents that contain the phrase animal. Documents that contain the word dog are not returned.

Related Topics

You can browse your thesaurus using procedures in the CTX_THES package.
See Also: For more information on browsing the top terms in your thesaurus, see CTX_THES.PT in Chapter 12, "CTX_THES Package".
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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>term*n</code></td>
<td>Returns documents that contain <code>term</code>. Calculates score by multiplying the raw score of <code>term</code> by <code>n</code>, where <code>n</code> is a number from 0.1 to 10.</td>
</tr>
</tbody>
</table>

### Examples

You have a collection of sports articles. You are interested in the articles about soccer, in particular Brazilian soccer. It turns out that a regular query on `soccer or Brazil` returns many high ranking articles on US soccer. To raise the ranking of the articles on Brazilian soccer, you can issue the following query:

'`soccer or Brazil*3'`  

Table 3–2 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.
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.

<table>
<thead>
<tr>
<th></th>
<th>soccer</th>
<th>Brazil</th>
<th>soccer or Brazil</th>
<th>soccer or Brazil*3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>20</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>30</td>
<td>30</td>
<td>90</td>
</tr>
<tr>
<td>C</td>
<td>50</td>
<td>20</td>
<td>50</td>
<td>60</td>
</tr>
</tbody>
</table>

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.
Wildcard characters can be used in query expressions to expand word searches into pattern searches. The wildcard characters are:

<table>
<thead>
<tr>
<th>Wildcard Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>The percent wildcard can appear any number of times at any part of the search term. The search term will be expanded into an equivalence list of terms. The list consists of all terms in the index that match the wildcarded term, with zero or more characters in place of the percent character.</td>
</tr>
<tr>
<td>_</td>
<td>The underscore wildcard specifies a single position in which any character can occur.</td>
</tr>
</tbody>
</table>

**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'
```

For all words that end with `ing`, you can issue:

```
'ing'
```
You can also combine left-truncated and right-truncated searches to create double-truncated searches. The following query finds all documents that contain words that contain the substring `%benz%`

```
' %benz%
```

**Improving Wildcard Query Performance**

You can improve wildcard query performance by adding a substring or prefix index.

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

When your wildcard queries are right-truncated, you can improve performance by creating a prefix index. A prefix index improves query performance for wildcard searches such as `to%`.

**See Also:** For more information about creating substring and prefix indexes, see "BASIC_WORDLIST” in Chapter 2.
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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expression WITHIN section</code></td>
<td>Searches for <code>expression</code> within the pre-defined zone, field, or attribute section.</td>
</tr>
<tr>
<td></td>
<td>If section is a zone, <code>expression</code> can contain one or more WITHIN operators (nested WITHIN) whose section is a zone or special section.</td>
</tr>
<tr>
<td></td>
<td>If section is a field or attribute section, <code>expression</code> cannot contain another WITHIN operator.</td>
</tr>
<tr>
<td><code>expression WITHIN SENTENCE</code></td>
<td>Searches for documents that contain <code>expression</code> within a sentence. Specify an AND or NOT query for <code>expression</code>.</td>
</tr>
<tr>
<td></td>
<td>The expression can contain one or more WITHIN operators (nested WITHIN) whose section is a zone or special section.</td>
</tr>
<tr>
<td><code>expression WITHIN PARAGRAPH</code></td>
<td>Searches for documents that contain <code>expression</code> within a paragraph. Specify an AND or NOT query for <code>expression</code>.</td>
</tr>
<tr>
<td></td>
<td>The expression can contain one or more WITHIN operators (nested WITHIN) whose section is a zone or special section.</td>
</tr>
</tbody>
</table>

WITHIN Limitations

The WITHIN operator has the following limitations:

- You cannot embed the WITHIN clause in a phrase. For example, you cannot write: `term1 WITHIN section term2`
Since WITHIN is a reserved word, you must escape the word with braces to search on it.

**WITHIN Operator Examples**

**Querying Within Zone Sections**
To find all the documents that contain the term *San Francisco* within the section *Headings*, write your query as follows:

'San Francisco WITHIN Headings'

To find all the documents that contain the term *sailing* and contain the term *San Francisco* within the section *Headings*, write your query in one of two ways:

'(San Francisco WITHIN Headings) and sailing'

'sailing and San Francisco WITHIN Headings'

**Compound Expressions with WITHIN**
To find all documents that contain the terms *dog* and *cat* within the same section *Headings*, write your query as follows:

'(dog and cat) WITHIN Headings'

This query is logically different from:

'dog WITHIN Headings and cat WITHIN Headings'

This query finds all documents that contain *dog* and *cat* where the terms *dog* and *cat* are in *Headings* sections, regardless of whether they occur in the same *Headings* section or different sections.

**Near with WITHIN**
To find all documents in which *dog* is near *cat* within the section *Headings*, write your query as follows:

'dog near cat WITHIN Headings'

---

**Note:** The near operator has higher precedence than the WITHIN operator so braces are not necessary in this example. This query is equivalent to (dog near cat) WITHIN Headings.
Nested WITHIN Queries

You can nest the within operator to search zone sections within zone sections.

For example, assume that a document set had the zone section AUTHOR nested within the zone BOOK section. You write a nested WITHIN query to find all occurrences of scott within the AUTHOR section of the BOOK section as follows:

'(scott WITHIN AUTHOR) WITHIN BOOK'

Querying Within Field Sections

The syntax for querying within a field section is the same as querying within a zone section. The syntax for most of the examples given in the previous section, "Querying Within Zone Sections", apply to field sections.

However, field sections behave differently from zone sections in terms of

- Visibility: You can make text within a field section invisible.
- Repeatability: WITHIN queries cannot distinguish repeated field sections.
- Nestability: You cannot issue a nested WITHIN query with a field section.

The following sections describe these differences.

Visible Flag in Field Sections

When a field section is created with the visible flag set to FALSE in CTX_DDL.ADD_FIELD_SECTION, the text within a field section can only be queried using the WITHIN operator.

For example, assume that TITLE is a field section defined with visible flag set to FALSE. Then the query dog without the WITHIN operator will not find a document containing:

<TITLE>The dog</TITLE> I like my pet.

To find such a document, you can use the WITHIN operator as follows:

'dog WITHIN TITLE'

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

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

WITHIN queries cannot distinguish repeated field sections in a document. For example, consider the document with the repeated section `<author>`:

```
<author> Charles Dickens </author>
<author> Martin Luther King </author>
```

Assuming that `<author>` is defined as a field section, a query such as `(charles and martin) within author` returns the document, even though these words occur in separate tags.

To have WITHIN queries distinguish repeated sections, define the sections as zone sections.

Nested Field Sections

You cannot issue a nested WITHIN query with field sections. Doing so raises an error.

Querying Within Sentence or Paragraphs

Querying within sentence or paragraph boundaries is useful to find combinations of words that occur in the same sentence or paragraph. To query sentence or paragraphs, you must first add the special section to your section group before you index. You do so with `CTX_DDL.ADD_SPECIAL_SECTION`.

To find documents that contain `dog` and `cat` within the same sentence:

```
'(dog and cat) WITHIN SENTENCE'
```

To find documents that contain `dog` and `cat` within the same paragraph:

```
'(dog and cat) WITHIN PARAGRAPH'
```

To find documents that contain sentences with the word `dog` but not `cat`:

```
'(dog not cat) WITHIN SENTENCE'
```

Querying Within Attribute Sections

You can query within attribute sections when you index with either `XML_SECTION_GROUP` or `AUTO_SECTION_GROUP` as your section group type.

Assume you have an XML document as follows:

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

  ```xml
  <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:

  ```xml
  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:

  ```xml
  <book title="Tale of Two Cities">It was the best of times.</book>
  <book title="Of Human Bondage">The sky broke dull and gray.</book>
  ```

  Assume that book is a zone section and book@author is an attribute section. Consider the query:
'(Tale and Bondage) WITHIN book@author'

This query does not hit the document, because tale and bondage are in different occurrences of the attribute section book@author.

Notes

Section Names
The WITHIN operator requires you to know the name of the section you search. A list of defined sections can be obtained using the CTX_SECTIONS or CTX_USER_SECTIONS views.

Section Boundaries
For special and zone sections, the terms of the query must be fully enclosed in a particular occurrence of the section for the document to satisfy the query. This is not a requirement for field sections.

For example, consider the query where bold is a zone section:

'(dog and cat) WITHIN bold'

This query finds:

<B>dog cat</B>

but it does not find:

<B>dog</B><B>cat</B>

This is because dog and cat must be in the same bold section.

This behavior is especially useful for special sections, where

'(dog and cat) WITHIN sentence'

means find dog and cat within the same sentence.

Field sections on the other hand are meant for non-repeating, embedded metadata such as a title section. Queries within field sections cannot distinguish between occurrences. All occurrences of a field section are considered to be parts of a single section. For example, the query:

(dog and cat) WITHIN title

can find a document like this:
<TITLE>dog</TITLE><TITLE>cat</TITLE>

In return for this field section limitation and for the overlap and nesting limitations, field section queries are generally faster than zone section queries, especially if the section occurs in every document, or if the search term is common.
This chapter describes the special characters that can be used in Text queries. In addition, it provides a list of the words and characters that Oracle Text treats as reserved words and characters.

The following topics are covered in this chapter:

- **Grouping Characters**
- **Escape Characters**
- **Reserved Words and Characters**

### Grouping Characters

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

<table>
<thead>
<tr>
<th>Grouping Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>The parentheses characters serve to group terms and operators found between the characters</td>
</tr>
<tr>
<td>[ ]</td>
<td>The bracket characters serve to group terms and operators found between the characters; however, they prevent penetrations for the expansion operators (fuzzy, soundex, stem).</td>
</tr>
</tbody>
</table>

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.
Escape Characters

For example, the open parenthesis indicates the beginning of a group. The first close parenthesis encountered is the end of the group. Any open parentheses encountered before the close parenthesis indicate nested groups.

Escape Characters

To query on words or symbols that have special meaning to query expressions such as `and & or | accum`, you must escape them. There are two ways to escape characters in a query expression:

<table>
<thead>
<tr>
<th>Escape Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>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.</td>
</tr>
<tr>
<td>\</td>
<td>Use the backslash character to escape a single character or symbol. Only the character immediately following the backslash is escaped. For example, a query of <code>blue\green</code> matches <code>blue-green</code> and <code>blue green</code>.</td>
</tr>
</tbody>
</table>

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'`

In the second example, the query matches `high-voltage` or `high voltage`.

**Note:** If you use braces to escape an individual character within a word, the character is escaped, but the word is broken into three tokens.

For example, a query written as `high\-voltage` searches for `high-voltage`, with the space on either side of the hyphen.
Querying Escape Characters

The open brace { signals the beginning of the escape sequence, and the closed brace } indicates the end of the sequence. Everything between the opening brace and the closing brace is part of the escaped query expression (including any open brace characters). To include the close brace character in an escaped query expression, use } }.

To escape the backslash escape character, use \\\\.

Reserved Words and Characters

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

<table>
<thead>
<tr>
<th>Reserved Word</th>
<th>Reserved Character</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABOUT</td>
<td>(none)</td>
<td>ABOUT</td>
</tr>
<tr>
<td>ACCUM</td>
<td>,</td>
<td>Accumulate</td>
</tr>
<tr>
<td>AND</td>
<td>&amp;</td>
<td>And</td>
</tr>
<tr>
<td>BT</td>
<td>(none)</td>
<td>Broader Term</td>
</tr>
<tr>
<td>BTG</td>
<td>(none)</td>
<td>Broader Term Generic</td>
</tr>
<tr>
<td>BTI</td>
<td>(none)</td>
<td>Broader Term Instance</td>
</tr>
<tr>
<td>BTP</td>
<td>(none)</td>
<td>Broader Term Partitive</td>
</tr>
<tr>
<td>FUZZY</td>
<td>?</td>
<td>fuzzy</td>
</tr>
<tr>
<td></td>
<td>{ }</td>
<td>escape characters (multiple)</td>
</tr>
<tr>
<td></td>
<td>\</td>
<td>escape character (single)</td>
</tr>
<tr>
<td></td>
<td>( )</td>
<td>grouping characters</td>
</tr>
<tr>
<td></td>
<td>[ ]</td>
<td>grouping characters</td>
</tr>
<tr>
<td>HASPATH</td>
<td>(none)</td>
<td>HASPATH</td>
</tr>
<tr>
<td>INPATH</td>
<td>(none)</td>
<td>INPATH</td>
</tr>
<tr>
<td>MDATA</td>
<td>(none)</td>
<td>MDATA</td>
</tr>
<tr>
<td>MINUS</td>
<td>-</td>
<td>MINUS</td>
</tr>
<tr>
<td>NEAR</td>
<td>;</td>
<td>NEAR</td>
</tr>
<tr>
<td>NOT</td>
<td>~</td>
<td>NOT</td>
</tr>
</tbody>
</table>
### Reserved Words and Characters

<table>
<thead>
<tr>
<th>Reserved Word</th>
<th>Reserved Character</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT</td>
<td>(none)</td>
<td>Narrower Term</td>
</tr>
<tr>
<td>NTG</td>
<td>(none)</td>
<td>Narrower Term Generic</td>
</tr>
<tr>
<td>NTI</td>
<td>(none)</td>
<td>Narrower Term Instance</td>
</tr>
<tr>
<td>NTP</td>
<td>(none)</td>
<td>Narrower Term Partitive</td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td>OR</td>
</tr>
<tr>
<td>PT</td>
<td>(none)</td>
<td>Preferred Term</td>
</tr>
<tr>
<td>RT</td>
<td>(none)</td>
<td>Related Term</td>
</tr>
<tr>
<td>(none)</td>
<td>$</td>
<td>stem</td>
</tr>
<tr>
<td>(none)</td>
<td>!</td>
<td>soundex</td>
</tr>
<tr>
<td>SQE</td>
<td>(none)</td>
<td>Stored Query Expression</td>
</tr>
<tr>
<td>SYN</td>
<td>(none)</td>
<td>Synonym</td>
</tr>
<tr>
<td>(none)</td>
<td>&gt;</td>
<td>threshold</td>
</tr>
<tr>
<td>TR</td>
<td>(none)</td>
<td>Translation Term</td>
</tr>
<tr>
<td>TRSYN</td>
<td>(none)</td>
<td>Translation Term Synonym</td>
</tr>
<tr>
<td>TT</td>
<td>(none)</td>
<td>Top Term</td>
</tr>
<tr>
<td>(none)</td>
<td>*</td>
<td>weight</td>
</tr>
<tr>
<td>(none)</td>
<td>%</td>
<td>wildcard character (multiple)</td>
</tr>
<tr>
<td>(none)</td>
<td>_</td>
<td>wildcard character (single)</td>
</tr>
<tr>
<td>WITHIN</td>
<td>(none)</td>
<td>WITHIN</td>
</tr>
</tbody>
</table>
This chapter provides information for using the CTX_ADM PL/SQL package.

CTX_ADM contains the following stored procedures:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECOVER</td>
<td>Cleans up database objects for deleted Text tables.</td>
</tr>
<tr>
<td>SET_PARAMETER</td>
<td>Sets system-level defaults for index creation.</td>
</tr>
</tbody>
</table>

**Note:** Only the CTXSYS user can use the procedures in CTX_ADM.
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;
```
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_OUTPUT files)
- ctx_doc_key_type (default input key type for CTX_DOC procedures)
- file_access_role
- default_datastore (default datastore preference)
- default_filter_file (default filter preference for data stored in files)
- default_filter_text (default text filter preference)
- default_filter_binary (default binary filter preference)
- default_section_html (default html section group preference)
- default_section_xml (default xml section group preference)
- default_section_text (default text section group preference)
- default_lexer (default lexer preference)
- default_wordlist (default wordlist preference)
- default_stoplist (default stoplist preference)
- default_storage (default storage preference)
- default_ctxcat_lexer
- default_ctxcat_stoplist
- default_ctxcat_storage
SET_PARAMETER

- default_ctxcat_wordlist
- default_ctxrule_lexer
- default_ctxrule_stoplist
- default_ctxrule_storage
- default_ctxrule_wordlist

**See Also:** To learn more about the default values for these parameters, see "System Parameters" in Chapter 2.

**param_value**
Specify the value to assign to the parameter. For `max_index_memory` and `default_index_memory`, the value you specify must have the following syntax:

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

where K stands for kilobytes, M stands for megabytes, and G stands for gigabytes. For each of the other parameters, specify the name of a preference to use as the default for indexing.

**Example**
```
begin
    ctx_adm.set_parameter('default_lexer', 'my_lexer');
end;
```
This chapter provides reference information for using the CTX_CLS PL/SQL package. This package enables you to perform document classification.

**See Also:** The Oracle Text Application Developer’s Guide for more on document classification

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLUSTERING</td>
<td>Generates clusters for a document collection.</td>
</tr>
</tbody>
</table>
Use this procedure to generate query rules that select document categories. You must supply a training set consisting of categorized documents. Documents can be in any format supported by Oracle Text and must belong to one or more categories. This procedure generates the queries that define the categories and then writes the results to a table.

You must also have a document table and a category table. The category table must contain at least two categories.

For example, your document and category tables can be defined as:

```sql
create table trainingdoc(
    docid number primary key,
    text varchar2(4000));

create table category (docid trainingdoc(docid),
categoryid number);
```

You can use one of two syntaxes depending on the classification algorithm you need. The query compatible syntax uses the RULE_CLASSIFIER preference and generates rules as query strings. The support vector machine syntax uses the SVM_CLASSIFIER preference and generates rules in binary format. The SVM_CLASSIFIER is good for high classification accuracy, but because its rules are generated in binary format, they cannot be examined like the query strings generated with the RULE_CLASSIFIER. Note that only those document ids that appear in both the document table and the category table will impact RULE_CLASSIFIER and SVM_CLASSIFIER learning.

The CTX_CLS.TRAIN procedure requires that your document table have an associated context index. For best results, the index should be synchronized before running this procedure. SVM_CLASSIFIER syntax enables the use of an unpopulated context index, while query-compatible syntax requires that the context index be populated.

**See Also:** The Oracle Text Application Developer’s Guide for more on document classification.
Query Compatible Syntax

The following syntax generates query-compatible rules and is used with the `RULE_CLASSIFIER` preference. Use this syntax and preference when different categories are separated from others by several key words. An advantage of generating your rules as query strings is that you can easily examine the generated rules. This is different from generating SVM rules, which are in binary format.

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

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

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

**cattab**
Specify the name of the category table. You must have SELECT privilege on this table.

**catdocid**
Specify the name of the document id column in the category table. The document ids in this table must also exist in the document table. This column must a NUMBER.

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

**restab**
Specify the name of the result table. You must have INSERT privilege on this table.
**rescatid**
Specify the name of the category ID column in the result table. This column must be a NUMBER.

**resquery**
Specify the name of the query column in the result table. This column must be VARCHAR2, CHAR CLOB, NVARCHAR2, or NCHAR.

The queries generated in this column connect terms with AND or NOT operators, such as:

‘T1 & T2 ~ T3’

Terms can also be theme tokens and be connected with the ABOUT operator, such as:

‘about(T1) & about(T2) ~ about(T3)’

Generated rules also support WITHIN queries on field sections.

**resconfid**
Specify the name of the confidence column in the result table. This column contains the estimated probability from training data that a document is relevant if that document satisfies the query.

**preference_name**
Specify the name of the preference. For classifier types and attributes, see "Classifier Types" in Chapter 2, "Oracle Text Indexing Elements".

**Syntax for Support Vector Machine Rules**
The following syntax generates support vector machine (SVM) rules with the SVM_CLASSIFIER preference. This preference generates rules in binary format. Use this syntax when your application requires high classification accuracy.

```sql
CTX_CLS.TRAIN(
    index_name in varchar2,
    docid      in varchar2,
    cattab     in varchar2,
    catdocid   in varchar2,
    catid      in varchar2,
    restab     in varchar2,
    preference_name in varchar2);
```

**index_name**
Specify the name of the text index.
**docid**
Specify the name of docid column in document table.

**cattab**
Specify the name of category table.

**catdocid**
Specify the name of docid column in category table.

**catid**
Specify the name of category ID column in category table.

**restab**
Specify the name of result table.

The result table has the following format:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT_ID</td>
<td>NUMBER</td>
<td>The ID of the category.</td>
</tr>
<tr>
<td>TYPE</td>
<td>NUMBER(3) NOT NULL</td>
<td>0 for the actual rule or catid; 1 for other.</td>
</tr>
<tr>
<td>RULE</td>
<td>BLOB</td>
<td>The returned rule.</td>
</tr>
</tbody>
</table>

**preference_name**
Specify the name of user preference. For classifier types and attributes, see "Classifier Types" in Chapter 2, "Oracle Text Indexing Elements".

**Example**

The CTX_CLS.TRAIN procedure is used in supervised classification. For an extended example, see the Oracle Text Application Developer’s Guide.
CLUSTERING

Use this procedure to cluster a collection of documents. A cluster is a group of documents similar to each other in content. Clustering is also known as unsupervised classification.

Given a set of documents, this procedure assigns each document into a cluster according to the similarity with documents already in the cluster. The result is that documents in a cluster are more similar to one another than documents across different clusters. The more clusters produced, the greater the accuracy and quality of each cluster; however, producing more clusters requires more computing time.

Cluster output may be flat or hierarchical. Hierarchical clustering affords greater specificity of each cluster; however, it may require more computing power. In the case where you want to produce only a few clusters, non-hierarchical clustering may suffice.

See Also: For more information about clustering, see "Cluster Types" in Chapter 2, "Oracle Text Indexing Elements", as well as the Oracle Text Application Developer’s Guide.

A clustering result set is composed of document assignments and cluster descriptions. The document assignment result set contains information about the cluster to which the procedure assigned a document, and how similar the document is to the assigned cluster. This result set contains document identification, cluster identification, and similarity score between the cluster and assigned document.

The cluster description result set contains information about what topic a generated cluster is about. This result set contains cluster identification, cluster description text, suggested cluster label, number of documents assigned, and a quality score of the cluster.

There are two versions of this procedure: one with a table result set, and one with an in-memory result set.

Syntax: Table Result Set

```sql
ctx_cls.clustering (
    index_name  IN VARCHAR2,
    docid       IN VARCHAR2,
    doctab_name IN VARCHAR2,
    clstab_name IN VARCHAR2,
)"
pref_name   IN VARCHAR2  DEFAULT NULL
);

index_name
Specify the name of the context index on collection table.

docid
Specify the name of document ID column of the collection table.

doctab_name
Specify the name of document assignment table. This procedure creates the table with the following structure:

doc_assign(
   docid number,
   clusterid number,
   score number
);

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCID</td>
<td>Document ID to identify document.</td>
</tr>
<tr>
<td>CLUSTERID</td>
<td>ID of the cluster the document is assigned to. If CLUSTERID is -1, then the cluster contains &quot;miscellaneous&quot; documents; for example, documents that cannot be assigned to any other cluster category.</td>
</tr>
<tr>
<td>SCORE</td>
<td>The associated score between the document and cluster.</td>
</tr>
</tbody>
</table>

If you require more columns, you can create the table before you call this procedure.

clstab_name
Specify the name of the cluster description table. This procedure creates the table with the following structure:

cluster_desc(
   clusterid NUMBER,
   descriptor VARCHAR2(4000),
   label VARCHAR2(200),
   size NUMBER,
   quality_score NUMBER,
   parent NUMBER
);
If you require more columns, you can create the table before you call this procedure.

**pref_name**
Specify the name of the preference.

### Syntax: In-Memory Result Set

You can put the result set into in-memory structures for better performance. Two in-memory tables are defined in CTX_CLS package for document assignment and cluster description respectively.

```sql
CTX_CLS.CLUSTERING(
    index_name     IN VARCHAR2,
    docid          IN VARCHAR2,
    dids           IN DOCID_TAB,
    doctab_name    IN OUT NOCOPY DOC_TAB,
    clstab_name    IN OUT NOCOPY CLUSTER_TAB,
    pref_name      IN VARCHAR2  DEFAULT NULL
);
```

**index_name**
Specify the name of context index on the collection table.

**docid**
Specify the document id column of the collection table.

**dids**
Specify the name of the in-memory docid_tab.

```sql
TYPE docid_tab IS TABLE OF number INDEX BY BINARY_INTEGER;
```
doctab_name
Specify name of the document assignment in-memory table. This table is defined as follows:

```sql
TYPE doc_rec IS RECORD (  
    docid NUMBER,  
    clusterid NUMBER,  
    score NUMBER  
)
```

```sql
TYPE doc_tab IS TABLE OF doc_rec INDEX BY BINARY_INTEGER;
```

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCID</td>
<td>Document ID to identify document.</td>
</tr>
<tr>
<td>CLUSTERID</td>
<td>ID of the cluster the document is assigned to. If CLUSTERID is -1, then the cluster contains &quot;miscellaneous&quot; documents; for example, documents that cannot be assigned to any other cluster category.</td>
</tr>
<tr>
<td>SCORE</td>
<td>The associated score between the document and cluster.</td>
</tr>
</tbody>
</table>

cls_tab
Specify the name of cluster description in-memory table

```sql
TYPE cluster_rec IS RECORD (  
    clusterid NUMBER,  
    descript VARCHAR2(4000),  
    label VARCHAR2(200),  
    sze NUMBER,  
    quality_score NUMBER,  
    parent NUMBER  
);
```

```sql
TYPE cluster_tab IS TABLE OF cluster_rec INDEX BY BINARY_INTEGER;
```

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLUSTERID</td>
<td>Cluster ID to identify cluster. If CLUSTERID is -1, then the cluster contains &quot;miscellaneous&quot; documents; for example, documents that cannot be assigned to any other cluster category.</td>
</tr>
<tr>
<td>DESCRIPT</td>
<td>String to describe the cluster.</td>
</tr>
<tr>
<td>LABEL</td>
<td>A suggested label for the cluster.</td>
</tr>
<tr>
<td>SZE</td>
<td>Number of documents assigned to this cluster.</td>
</tr>
</tbody>
</table>
### pref_name
Specify the name of the preference. For cluster types and attributes, see "Cluster Types" in Chapter 2, "Oracle Text Indexing Elements".

### Example

**See Also:** The Oracle Text Application Developer’s Guide for an example of using clustering.
This chapter provides reference information for using the CTX_DDL PL/SQL package to create and manage the preferences, section groups, and stoplists required for Text indexes.

CTX_DDL contains the following stored procedures and functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_ATTR_SECTION</td>
<td>Adds an attribute section to a section group.</td>
</tr>
<tr>
<td>ADD_FIELD_SECTION</td>
<td>Creates a field section and assigns it to the specified section group.</td>
</tr>
<tr>
<td>ADD_INDEX</td>
<td>Adds an index to a catalog index preference.</td>
</tr>
<tr>
<td>ADD_MDATA</td>
<td>Changes the MDATA value of a document.</td>
</tr>
<tr>
<td>ADD_MDATA_SECTION</td>
<td>Adds an MDATA metadata section to a document.</td>
</tr>
<tr>
<td>ADD_SPECIAL_SECTION</td>
<td>Adds a special section to a section group.</td>
</tr>
<tr>
<td>ADD_STOPCLASS</td>
<td>Adds a stopclass to a stoplist.</td>
</tr>
<tr>
<td>ADD_STOP_SECTION</td>
<td>Adds a stop section to an automatic section group.</td>
</tr>
<tr>
<td>ADD_STOPTHEME</td>
<td>Adds a stoptheme to a stoplist.</td>
</tr>
<tr>
<td>ADD_STOPWORD</td>
<td>Adds a stopword to a stoplist.</td>
</tr>
<tr>
<td>ADD_SUB_LEXER</td>
<td>Adds a sub-lexer to a multi-lexer preference.</td>
</tr>
<tr>
<td>ADD_ZONE_SECTION</td>
<td>Creates a zone section and adds it to the specified section group.</td>
</tr>
<tr>
<td>CREATE_INDEX_SET</td>
<td>Creates an index set for CTXCAT index types.</td>
</tr>
<tr>
<td>CREATE_POLICY</td>
<td>Create a policy to use with ORA:CONTAINS().</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>CREATE_PREFERENCE</td>
<td>Creates a preference in the Text data dictionary</td>
</tr>
<tr>
<td>CREATE_SECTION_GROUP</td>
<td>Creates a section group in the Text data dictionary</td>
</tr>
<tr>
<td>CREATE_STOPLIST</td>
<td>Creates a stoplist.</td>
</tr>
<tr>
<td>DROP_INDEX_SET</td>
<td>Drops an index set.</td>
</tr>
<tr>
<td>DROP_POLICY</td>
<td>Drops a policy.</td>
</tr>
<tr>
<td>DROP_PREFERENCE</td>
<td>Deletes a preference from the Text data dictionary</td>
</tr>
<tr>
<td>DROP_SECTION_GROUP</td>
<td>Deletes a section group from the Text data dictionary</td>
</tr>
<tr>
<td>DROP_STOPLIST</td>
<td>Drops a stoplist.</td>
</tr>
<tr>
<td>OPTIMIZE_INDEX</td>
<td>Optimize the index.</td>
</tr>
<tr>
<td>REMOVE_INDEX</td>
<td>Removes an index from a CTXCAT index preference</td>
</tr>
<tr>
<td>REMOVE_MDATA</td>
<td>Removes MDATA values from a document</td>
</tr>
<tr>
<td>REMOVE_SECTION</td>
<td>Deletes a section from a section group</td>
</tr>
<tr>
<td>REMOVE_STOPCLASS</td>
<td>Deletes a stopclass from a section group.</td>
</tr>
<tr>
<td>REMOVE_STOPTHEME</td>
<td>Deletes a stoptheme from a stoplist.</td>
</tr>
<tr>
<td>REMOVE_STOPWORD</td>
<td>Deletes a stopword from a section group.</td>
</tr>
<tr>
<td>REPLACE_INDEX_METADATA</td>
<td>Replaces metadata for local domain indexes</td>
</tr>
<tr>
<td>SET_ATTRIBUTE</td>
<td>Sets a preference attribute.</td>
</tr>
<tr>
<td>SYNC_INDEX</td>
<td>Synchronize index.</td>
</tr>
<tr>
<td>UNSET_ATTRIBUTE</td>
<td>Removes a set attribute from a preference.</td>
</tr>
<tr>
<td>UPDATE_POLICY</td>
<td>Updates a policy.</td>
</tr>
</tbody>
</table>
ADD_ATTR_SECTION

Adds an attribute section to an XML section group. This procedure is useful for defining attributes in XML documents as sections. This enables you to search XML attribute text with the WITHIN operator.

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

**Syntax**

```sql
CTX_DDL.ADD_ATTR_SECTION(
  group_name     in    varchar2,
  section_name   in    varchar2,
  tag            in    varchar2);
```

*group_name*
Specify the name of the XML section group. You can add attribute sections only to XML section groups.

*section_name*
Specify the name of the attribute section. This is the name used for WITHIN queries on the attribute text.

The section name you specify cannot contain the colon (:), comma (,), or dot (.) characters. The section name must also be unique within group_name. Section names are case-insensitive.

Attribute section names can be no more than 64 bytes long.

*tag*
Specify the name of the attribute in tag@attr form. This parameter is case-sensitive.

**Examples**

Consider an XML file that defines the BOOK tag with a TITLE attribute as follows:

```xml
<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:

```sql
begin
    ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
    ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'BOOK#TITLE');
end;
```

When you define the TITLE attribute section as such and index the document set, you can query the XML attribute text as follows:

'Cities within booktitle'
ADD_FIELD_SECTION

Creates a field section and adds the section to an existing section group. This enables field section searching with the WITHIN operator.

Field sections are delimited by start and end tags. By default, the text within field sections are indexed as a sub-document separate from the rest of the document.

Unlike zone sections, field sections cannot nest or overlap. As such, field sections are best suited for non-repeating, non-overlapping sections such as TITLE and AUTHOR markup in email- or news-type documents.

Because of how field sections are indexed, WITHIN queries on field sections are usually faster than WITHIN queries on zone sections.

Syntax

```sql
CTX_DDL.ADD_FIELD_SECTION(
    group_name     in    varchar2,
    section_name   in    varchar2,
    tag            in    varchar2,
    visible        in    boolean default FALSE
);
```

group_name
Specify the name of the section group to which section_name is added. You can add up to 64 field sections to a single section group. Within the same group, section zone names and section field names cannot be the same.

section_name
Specify the name of the section to add to the group_name. You use this name to identify the section in queries. Avoid using names that contain non-alphanumeric characters such as _, since these characters must be escaped in queries. Section names are case-insensitive.

Within the same group, zone section names and field section names cannot be the same. The terms Paragraph and Sentence are reserved for special sections.

Section names need not be unique across tags. You can assign the same section name to more than one tag, making details transparent to searches.
**tag**
Specify the tag which marks the start of a section. For example, if the tag is `<H1>`, specify H1. The start tag you specify must be unique within a section group.

If `group_name` is an `HTML_SECTION_GROUP`, you can create field sections for the META tag's `NAME/CONTENT` attribute pairs. To do so, specify `tag` as `meta@namevalue` where `namevalue` is the value of the NAME attribute whose CONTENT attribute is to be indexed as a section. Refer to the example.

Oracle Text knows what the end tags look like from the `group_type` parameter you specify when you create the section group.

**visible**
Specify `TRUE` to make the text visible within rest of document.

By default the `visible` flag is `FALSE`. This means that Oracle Text indexes the text within field sections as a sub-document separate from the rest of the document. However, you can set the visible flag to `TRUE` if you want text within the field section to be indexed as part of the enclosing document.

**Examples**

### Visible and Invisible Field Sections

The following 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`:

```sql
begin
txt_ddl.create_section_group('basicgroup', 'BASIC_SECTION_GROUP');
txt_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:

```sql
begin
txt_ddl.add_field_section('basicgroup', 'Author', 'A', TRUE);
end;
```
Creating Sections for `<META>` Tags

When you use the `HTML_SECTION_GROUP`, you can create sections for `META` tags.

Consider an HTML document that has a `META` tag as follows:

```html
<META NAME="author" CONTENT="ken">
```

To create a field section that indexes the `CONTENT` attribute for the `<META NAME="author">` tag:

```sql
begin
ctx_ddl.create_section_group('myhtmlgroup', 'HTML_SECTION_GROUP');
ctx_ddl.add_field_section('myhtmlgroup', 'author', 'META@AUTHOR');
end
```

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

'ken WITHIN author'

Limitations

Nested Sections

Field sections cannot be nested. For example, if you define a field section to start with `<TITLE>` and define another field section to start with `<FOO>`, the two sections cannot be nested as follows:

```html
<TITLE> dog <FOO> cat </FOO> </TITLE>
```

To work with nested sections 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:

```html
<TITLE> cat </TITLE>
<TITLE> dog </TITLE>
```

The query (dog and cat) within title returns the document, even though these words occur in different sections.

To have `WITHIN` queries distinguish repeated sections, define them as zone sections.
Related Topics

WITHIN operator in Chapter 3, "CONTAINS Query Operators".
"Section Group Types" in Chapter 2, "Oracle Text Indexing Elements".
CREATE_SECTION_GROUP
ADD_ZONE_SECTION
ADD_SPECIAL_SECTION
REMOVE_SECTION
DROP_SECTION_GROUP
Use this procedure to add a sub-index to a catalog index preference. You create this preference by naming one or more columns in the base table.

Since you create sub-indexes to improve the response time of structured queries, the column you add should be used in the `structured_query` clause of the `CATSEARCH` operator at query-time.

**Syntax**

```
CTX_DDL.ADD_INDEX(set_name in varchar2,
column_list varchar2,
storage_clause varchar2);
```

- **set_name**
  Specify the name of the index set.

- **column_list**
  Specify a comma separated list of columns to index. At index time, any column listed here cannot have a NULL value in any row in the base table. If any row is NULL during indexing and error is raised.

  You must always ensure that your columns have non-NULL values before and after indexing.

- **storage_clause**
  Specify a storage clause.

**Example**

Consider a table called `AUCTION` with the following schema:

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

Assume that queries on the table involve a mandatory text query clause and optional structured conditions on `category_id`. Results must be sorted based on `bid_close`.
You can create a catalog index to support the different types of structured queries a user might enter.

To create the indexes, first create the index set preference then add the required indexes to it:

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

Create the combined catalog index with `CREATE INDEX` as follows:

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

**Querying**

To query the title column for the word `pokemon`, you can issue regular and mixed queries as follows:

```sql
select * from AUCTION where CATSEARCH(title, 'pokemon',NULL)> 0;
select * from AUCTION where CATSEARCH(title, 'pokemon', 'category_id=99 order by bid_close desc')> 0;
```
ADD_MDATA

Use this procedure to change the metadata of a document that has been specified as an MDATA section. After this call, MDATA queries involving the named MDATA value will find documents with the given MDATA value.

There are two versions of CTX_DDL.ADD_MDATA: one for adding a single metadata value to a single rowid, and one for handing multiple values, multiple rowids, or both.

CTX_DDL.ADD_MDATA is transactional; it takes effect immediately in the calling session, can be seen only in the calling session, can be reversed with a ROLLBACK command, and must be committed to take permanent effect.

Use CTX_DDL.REMOVE_MDATA to remove metadata values from already-indexed documents. Only the owner of the index is allowed to call ADD_MDATA and REMOVE_MDATA.

Syntax

This is the syntax for adding a single value to a single rowid:

```sql
CTX_DDL.ADD_MDATA(
    idx_name           IN VARCHAR2,
    section_name       IN VARCHAR2,
    mdata_value        IN VARCHAR2,
    mdata_rowid        IN VARCHAR2,
    [part_name]        IN VARCHAR2
);
```

**idx_name**
Name of the text index that contains the named rowid.

**section_name**
Name of the MDATA section.

**mdata_value**
The metadata value to add to the document.

**mdata_rowid**
The rowid to which to add the metadata value.
[part_name]
Name of the index partition, if any. Must be provided for local partitioned indexes and must be NULL for global indexes.

This is the syntax for handling multiple values, multiple rowids, or both. This version is more efficient for large numbers of new values or rowids.

```
CTX_DDL.ADD_MDATA(
    idx_name           IN VARCHAR2,
    section_name       IN VARCHAR2,
    mdata_values       SYS.ODCIVARCHAR2LIST,
    mdata_rowids       SYS.ODCIRIDLIST,
    [part_name]        IN VARCHAR2
);
```

**idx_name**
Name of the text index that contains the named rowids.

**section_name**
Name of the MDATA section.

**mdata_values**
List of metadata values. If a metadata value contains a comma, the comma must be escaped with a backslash.

**mdata_rowids**
Rowids to which to add the metadata values.

[part_name]
Name of the index partition, if any. Must be provided for local partitioned indexes and must be NULL for global indexes.

**Example**

This example updates a single value:

```
SQL> select rowid from mytab where contains(text, 'MDATA(sec, value')>0;
No rows returned
SQL> exec ctx_ddl.add_mdata('my_index', 'sec', 'value', 'ABC');
SQL> select rowid from mytab where contains(text, 'MDATA(sec, value')>0;
<table>
<thead>
<tr>
<th>ROWID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
</tr>
</tbody>
</table>
```

This example updates multiple values:

```
SQL> exec ctx_ddl.add_mdata('my_index', 'sec', 'value', 'ABC', 'DEF');
SQL> select rowid from mytab where contains(text, 'MDATA(sec, value')>0;
<table>
<thead>
<tr>
<th>ROWID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
</tr>
<tr>
<td>DEF</td>
</tr>
</tbody>
</table>
```
begin
ctx_ddl.add_mdata('my_index', 'sec',
    sys.odcivarchar2list('value1','value2','value3'),
    sys.odciridlist('ABC','DEF'));
end;

This is equivalent to:
begin
    ctx_ddl.add_mdata('my_index', 'sec', 'value1', 'ABC');
    ctx_ddl.add_mdata('my_index', 'sec', 'value1', 'DEF');
    ctx_ddl.add_mdata('my_index', 'sec', 'value2', 'ABC');
    ctx_ddl.add_mdata('my_index', 'sec', 'value2', 'DEF');
    ctx_ddl.add_mdata('my_index', 'sec', 'value3', 'ABC');
    ctx_ddl.add_mdata('my_index', 'sec', 'value3', 'DEF');
end;

Notes

If a rowid is not yet indexed, CTX_DDL.ADD_MDATA completes without error, but an error is logged in CTX_USER_INDEX_ERRORS.

Related Topics

See also “ADD_MDATA_SECTION” on page 7-14; “REMOVE_MDATA” on page 7-53; “MDATA” on page 3-27; as well as the Section Searching chapter of the Oracle Text Application Developer’s Guide.
Use this procedure to add an MDATA section, with an accompanying value, to an existing section group. MDATA sections cannot be added to Null Section groups, Path Section groups, or Auto Section groups.

Section values undergo a simplified normalization:

- Leading and trailing whitespace on the value is removed.
- The value is truncated to 64 bytes.
- The value is converted to upper case.
- The value is indexed as a single value; if the value consists of multiple words, it is not broken up.
- Case is preserved. If the document is dynamically generated, you can implement case-insensitivity by uppercasing MDATA values and making sure to search only in uppercase.

Use CTX_DDL.REMOVE_SECTION to remove sections.

Syntax

```
CTX_DDL.ADD_MDATA_SECTION(
    group_name    IN VARCHAR2,
    section_name  IN VARCHAR2,
    tag           IN VARCHAR2,
);
```

**group_name**

Name of the section group that will contain the MDATA section.

**section_name**

Name of the MDATA section.

**tag**

The value of the MDATA section. For example, if the section is `<AUTHOR>`, the value could be Cynthia Kadohata (author of the novel *The Floating World*). More than one tag can be assigned to a given MDATA section.
Example

This example creates an MDATA section called AUTHOR and gives it the value *Gordon Burn* (author of the novel *Alma*).

```java
ctx_ddl.create.section.group('htmgrep', 'HTML_SECTION_GROUP');
ctx_ddl.add_mdata_section('htmgrep', 'author', 'Gordon Burn');
```

Related Topics

See also "ADD_MDATA" on page 7-11; "REMOVE_MDATA" on page 7-53; "MDATA" on page 3-27; "CREATE_SECTION_GROUP" on page 7-38, as well as the Section Searching chapter of the *Oracle Text Application Developer’s Guide*. 
ADD_SPECIAL_SECTION

Adds a special section, either SENTENCE or PARAGRAPH, to a section group. This enables searching within sentences or paragraphs in documents with the WITHIN operator.

A special section in a document is a section which is not explicitly tagged like zone and field sections. The start and end of special sections are detected when the index is created. Oracle Text supports two such sections: paragraph and sentence.

The sentence and paragraph boundaries are determined by the lexer. For example, the lexer recognizes sentence and paragraph section boundaries as follows:

<table>
<thead>
<tr>
<th>Special Section</th>
<th>Boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SENTENCE</td>
<td>WORD/PUNCT/WHITESPACE</td>
</tr>
<tr>
<td></td>
<td>WORD/PUNCT/NEWLINE</td>
</tr>
<tr>
<td>PARAGRAPH</td>
<td>WORD/PUNCT/NEWLINE/WHITESPACE (indented paragraph)</td>
</tr>
<tr>
<td></td>
<td>WORD/PUNCT/NEWLINE/NEWLINE (block paragraph)</td>
</tr>
</tbody>
</table>

The punctuation, whitespace, and newline characters are determined by your lexer settings and can be changed.

If the lexer cannot recognize the boundaries, no sentence or paragraph sections are indexed.

Syntax

```
CTX_DDL.ADD_SPECIAL_SECTION(
    group_name IN VARCHAR2,
    section_name IN VARCHAR2);
```

group_name
Specify the name of the section group.

section_name
Specify SENTENCE or PARAGRAPH.
Example

The following code enables searching within sentences within HTML documents:

```sql
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`:

```sql
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:

```sql
begin
  ctx_ddl.create_section_group('nullgroup', 'NULL_SECTION_GROUP');
  ctx_ddl.add_special_section('nullgroup', 'SENTENCE');
end;
```

Related Topics

- `WITHIN` operator in Chapter 3, "CONTAINS Query Operators".
- "Section Group Types" in Chapter 2, "Oracle Text Indexing Elements".
- `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

```plaintext
CTX_DDL.ADD_STOPCLASS(
    stoplist_name  in   varchar2,
    stopclass      in   varchar2
);
```

- **stoplist_name**: Specify the name of the stoplist.
- **stopclass**: Specify the stopclass to be added to stoplist_name. Currently, only the NUMBERS class is supported.

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

Example

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

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

Related Topics

- CREATE_STOPLIST
- REMOVE_STOPCLASS
- DROP_STOPLIST
Adds a stop section to an automatic section group. Adding a stop section causes the automatic section indexing operation to ignore the specified section in XML documents.

**Note:** Adding a stop section causes no section information to be created in the index. However, the text within a stop section is always searchable.

Adding a stop section is useful when your documents contain many low information tags. Adding stop sections also improves indexing performance with the automatic section group.

The number of stop sections you can add is unlimited.

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

### Syntax

```sql
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`:

```sql
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:

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

Then the preceding 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`:

```sql
begin
ctx_ddl.add_stop_section('myauto', '(mydoc)fluff');
end;
```

Related Topics

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

CREATESECTIONGROUP
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

```sql
CTX_DDL.ADD_STOPTHEME(
    stoplist_name  in   varchar2,
    stoptheme      in   varchar2
);
```

**stoplist_name**
Specify the name of the stoplist.

**stoptheme**
Specify the stoptheme to be added to stoplist_name. The system normalizes the stoptheme you enter using the knowledge base. If the normalized theme is more than one theme, the system does not process your stoptheme. For this reason, Oracle recommends that you submit single stopthemes.

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

Example

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

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

Related Topics

- CREATE_STOPLIST
- REMOVE_STOPTHEME
- DROP_STOPLIST
- ABOUT operator in Chapter 3, "CONTAINS Query Operators".
ADD_STOPWORD

Use this procedure to add a single stopword to a stoplist.
To create a list of stopwords, you must call this procedure once for each word.

Syntax

```
CTX_DDL.ADD_STOPWORD(
    stoplist_name  in   varchar2,
    stopword       in   varchar2,
    language       in varchar2 default NULL
);
```

**stoplist_name**
Specify the name of the stoplist.

**stopword**
Specify the stopword to be added.

Language-specific stopwords must be unique across the other stopwords specific to the language. For example, it is valid to have a German `die` and an English `die` in the same stoplist.

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

**language**
Specify the language of `stopword` when the stoplist you specify with `stoplist_name` is of type MULTI_STOPLIST. You must specify the Globalization Support name or abbreviation of an Oracle Text-supported language.

To make a stopword active in multiple languages, specify ALL for this parameter. For example, defining ALL stopwords is useful when you have international documents that contain English fragments that need to be stopped in any language.

An ALL stopword is active in all languages. If you use the multi-lexer, the language-specific lexing of the stopword occurs, just as if it had been added multiple times in multiple specific languages.

Otherwise, specify NULL.
Example

**Single Language Stoplist**
The following example adds the stopwords *because, notwithstanding, nonetheless, and therefore* to the stoplist *mystop*:

```java
begin
ctx_ddl.add_stopword('mystop', 'because');
ctx_ddl.add_stopword('mystop', 'notwithstanding');
ctx_ddl.add_stopword('mystop', 'nonetheless');
ctx_ddl.add_stopword('mystop', 'therefore');
end;
```

**Multi-Language Stoplist**
The following example adds the German word *die* to a multi-language stoplist:

```java
begin
ctx_ddl.add_stopword('mystop', 'Die','german');
end;
```

---

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

---

**Adding An ALL Stopword**
The following adds the word *the* as an ALL stopword to the multi-language stoplist *globallist*:

```java
begin
ctx_ddl.add_stopword('globallist','the','ALL');
end;
```

---

**Related Topics**

CREATE_STOPLIST
REMOVE_STOPWORD
DROP_STOPLIST
ALTER INDEX in Chapter 1, "SQL Statements and Operators".
Appendix 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.

Restrictions

The following restrictions apply to using CTX_DDL.ADD_SUB_LEXER:

- The invoking user must be the owner of the multi-lexer or CTXSYS.
- The lexer_name parameter must name a preference which is a multi-lexer lexer.
- A lexer for default must be defined before the multi-lexer can be used in an index.
- The sub-lexer preference owner must be the same as multi-lexer preference owner.
- The sub-lexer preference must not be a multi-lexer lexer.
- A sub-lexer preference cannot be dropped while it is being used in a multi-lexer preference.
- CTX_DDL.ADD_SUB_LEXER records only a reference. The sub-lexer values are copied at create index time to index value storage.

Syntax

```sql
CTX_DDL.ADD_SUB_LEXER(
    lexer_name in varchar2,
    language  in varchar2,
    sub_lexer in varchar2,
    alt_value in varchar2 default null
);
```

**lexer_name**

Specify the name of the multi-lexer preference.

**language**

Specify the Globalization Support language name or abbreviation of the sub-lexer. For example, you can specify ENGLISH or EN for English.
The sub-lexer you specify with sublexer is used when the language column has a value case-insensitive equal to the Globalization Support name of abbreviation of language.

Specify DEFAULT to assign a default sub-lexer to use when the value of the language column in the base table is null, invalid, or unmapped to a sub-lexer. The DEFAULT lexer is also used to parse stopwords.

If a sub-lexer definition for language already exists, then it is replaced by this call.

**sublexer**
Specify the name of the sub-lexer to use for this language.

**alt_value**
Optionally specify an alternate value for language.

If you specify DEFAULT for language, you cannot specify an alt_value.

The alt_value is limited to 30 bytes and cannot be an Globalization Support language name, abbreviation, or DEFAULT.

**Example**

This example shows how to create a multi-language text table and how to set up the multi-lexer to index the table.

Create the multi-language table with a primary key, a text column, and a language column as follows:

```sql
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:

```sql
ctx_ddl.create_preference('english_lexer','basic_lexer');
ctx_ddl.set_attribute('english_lexer','index_themes','yes');
ctx_ddl.set_attribute('english_lexer','theme_language','english');

ctx_ddl.create_preference('german_lexer','basic_lexer');
ctx_ddl.set_attribute('german_lexer','composite','german');
ctx_ddl.set_attribute('german_lexer','mixed_case','yes');
```
ctx_ddl.set_attribute('german_lexer','alternate_spelling','german');

ctx_ddl.create_preference('japanese_lexer','japanese_vgram_lexer');

Create the multi-lexer preference:
ctx_ddl.create_preference('global_lexer', 'multi_lexer');

Since the stored documents are mostly English, make the English lexer the default:
ctx_ddl.add_sub_lexer('global_lexer', 'default', 'english_lexer');

Add the German and Japanese lexers in their respective languages. Also assume that the language column is expressed in ISO 639-2, so we add those as alternate values.

ctx_ddl.add_sub_lexer('global_lexer','german','german_lexer','ger');
ctx_ddl.add_sub_lexer('global_lexer','japanese','japanese_lexer','jpn');

Create the index globalx, specifying the multi-lexer preference and the language column in the parameters string as follows:
create index globalx on globaldoc(text) indextype is ctxsys.context
parameters ('lexer global_lexer language column lang');
ADD_ZONE_SECTION

Creates a zone section and adds the section to an existing section group. This enables zone section searching with the WITHIN operator.

Zone sections are sections delimited by start and end tags. The <B> and </B> tags in HTML, for instance, marks a range of words which are to be rendered in boldface.

Zone sections can be nested within one another, can overlap, and can occur more than once in a document.

Syntax

```
CTX_DDL.ADD_ZONE_SECTION(
    group_name     in    varchar2,
    section_name   in    varchar2,
    tag            in    varchar2
);
```

**group_name**
Specify the name of the section group to which section_name is added.

**section_name**
Specify the name of the section to add to the group_name. You use this name to identify the section in WITHIN queries. Avoid using names that contain non-alphanumeric characters such as _, since most of these characters are special must be escaped in queries. Section names are case-insensitive.

Within the same group, zone section names and field section names cannot be the same. The terms *Paragraph* and *Sentence* are reserved for special sections.

Section names need not be unique across tags. You can assign the same section name to more than one tag, making details transparent to searches.

**tag**
Specify the pattern which marks the start of a section. For example, if <H1> is the HTML tag, specify H1 for tag. The start tag you specify must be unique within a section group.

Oracle Text knows what the end tags look like from the group_type parameter you specify when you create the section group.
If \texttt{group\_name} is an \texttt{HTML\_SECTION\_GROUP}, you can create zone sections for the \texttt{META} tag's \texttt{NAME/CONTENT} attribute pairs. To do so, specify \texttt{tag} as \texttt{meta@namevalue} where \texttt{namevalue} is the value of the \texttt{NAME} attribute whose \texttt{CONTENT} attributes are to be indexed as a section. Refer to the example.

If \texttt{group\_name} is an \texttt{XML\_SECTION\_GROUP}, you can optionally qualify \texttt{tag} with a document type (root element) in the form \texttt{(doctype)tag}. Doing so makes \texttt{section\_name} sensitive to the XML document type declaration. Refer to the example.

**Examples**

**Creating HTML Sections**
The following code defines a section group called \texttt{htmgroup} of type \texttt{HTML\_SECTION\_GROUP}. It then creates a zone section in \texttt{htmgroup} called \texttt{headline} identified by the \texttt{<H1>} tag:

```sql
begin
ctx_ddl.create_section_group('htmgroup', 'HTML\_SECTION\_GROUP');
ctx_ddl.add_zone_section('htmgroup', 'heading', 'H1');
end;
```

After indexing with section group \texttt{htmgroup}, you can query within the heading section by issuing a query as follows:

```sql
'Oracle WITHIN heading'
```

**Creating Sections for \texttt{<META NAME>} Tags**
You can create zone sections for HTML \texttt{META} tags when you use the \texttt{HTML\_SECTION\_GROUP}.

Consider an HTML document that has a \texttt{META} tag as follows:

```html
<META NAME="author" CONTENT="ken">
```

To create a zone section that indexes all \texttt{CONTENT} attributes for the \texttt{META} tag whose \texttt{NAME} value is \texttt{author}:

```sql
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 \texttt{htmgroup}, you can query the document as follows:

```sql
'Oracle WITHIN author'
```
Creating Document Type Sensitive Sections (XML Documents Only)
You have an XML document set that contains the `<book>` tag declared for different document types (DTDs). You want to create a distinct book section for each document type.

Assume that `myDTDname` is declared as an XML document type as follows:

```xml
<!DOCTYPE myDTDname>
<myDTDname>
...
```

(Note: the DOCTYPE must match the top-level tag.)

Within `myDTDname`, 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:

```pl
begin
ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_zone_section('myxmlgroup', 'mybooksec', '(myDTDname)book');
end;
```

**Notes**

**Repeated Sections**
Zone sections can repeat. Each occurrence is treated as a separate section. For example, if `<H1>` denotes a heading section, they can repeat in the same documents as follows:

```html
<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:

```html
plain <B> bold <I> bold and italic </B> only italic </I> plain
```
Nested Sections
Zone sections can nest, including themselves as follows:
<TD> <TABLE><TD>nested cell</TD></TABLE></TD>

Using the WITHIN operator, you can write queries to search for text in sections within sections. For example, assume the BOOK1, BOOK2, and AUTHOR zone sections occur as follows in documents doc1 and doc2:
doc1:
<book1> <author>Scott Tiger</author> This is a cool book to read.</book1>
doc2:
<book2> <author>Scott Tiger</author> This is a great book to read.</book2>

Consider the nested query:
'(Scott within author) within book1'

This query returns only doc1.

Related Topics
WITHIN operator in Chapter 3, "CONTAINS Query Operators".
"Section Group Types" in Chapter 2, "Oracle Text Indexing Elements".
CREATE_SECTION_GROUP
ADD_FIELD_SECTION
ADD_SPECIAL_SECTION
REMOVE_SECTION
DROP_SECTION_GROUP
CREATE_INDEX_SET

Creates an index set for CTXCAT index types. You name this index set in the parameter clause of CREATE INDEX when you create a CTXCAT index.

Syntax

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

**set_name**
Specify the name of the index set. You name this index set in the parameter clause of CREATE INDEX when you create a CTXCAT index.
CREATE_POLICY

Creates a policy to use with the CTX_DOC.POLICY_* procedures and the ORA:CONTAINS function. ORA:CONTAINS is a function you use within an XPATH query expression with existsNode().

See Also: Oracle XML DB Developer’s Guide

Syntax

CTX_DDL.CREATE_POLICY(
    policy_name  IN VARCHAR2 DEFAULT NULL,
    filter       IN VARCHAR2 DEFAULT NULL,
    section_group IN VARCHAR2 DEFAULT NULL,
    lexer        IN VARCHAR2 DEFAULT NULL,
    stoplist     IN VARCHAR2 DEFAULT NULL,
    wordlist     IN VARCHAR2 DEFAULT NULL);

policy_name
Specify the name for the new policy.

filter
Specify the filter preference to use.

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

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

lexer
Specify the lexer preference to use. Your INDEX THEMES attribute must be disabled.

stoplist
Specify the stoplist to use.

wordlist
Specify the wordlist to use.
Example

Create mylex lexer preference named mylex.
begin
ctx_ddl.create_preference('mylex', 'BASIC_LEXER');
ctx_ddl.set_attribute('mylex', 'printjoins', '_-');
ctx_ddl.set_attribute('mylex', 'index_themes', 'NO');
ctx_ddl.set_attribute('mylex', 'index_text', 'YES');
end;

Create a stoplist preference named mystop.
begin
ctx_ddl.create_stoplist('mystop', 'BASIC_STOPLIST');
ctx_ddl.add_stopword('mystop', 'because');
ctx_ddl.add_stopword('mystop', 'nonetheless');
ctx_ddl.add_stopword('mystop', 'therefore');
end;

Create a wordlist preference named 'mywordlist'.
begin
ctx_ddl.create_preference('mywordlist', 'BASIC_WORDLIST');
ctx_ddl.set_attribute('mywordlist', 'FUZZY_MATCH', 'ENGLISH');
ctx_ddl.set_attribute('mywordlist', 'FUZZY_SCORE', '0');
ctx_ddl.set_attribute('mywordlist', 'FUZZY_NUMRESULTS', '5000');
ctx_ddl.set_attribute('mywordlist', 'SUBSTRING_INDEX', 'TRUE');
ctx_ddl.set_attribute('mywordlist', 'STEMMER', 'ENGLISH');
end;

exec ctx_ddl.create_policy('my_policy', NULL, NULL, 'mylex', 'mystop', 'mywordlist');

or

exec ctx_ddl.create_policy(policy_name => 'my_policy',
lexer => 'mylex',
stoplist => 'mystop',
wordlist => 'mywordlist');

Then you can issue the following existsNode() query with your own defined policy:

select id from xmltab
where existsNode(doc, '/book/chapter[ ora:contains(summary,"dog or cat", "my_
You can update your policy by doing:

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

You can drop your policy by doing:

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

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

Syntax

```
CTX_DDL.CREATE_PREFERENCE(preference_name  in varchar2,
                           object_name      in varchar2);
```

- **preference_name**: Specify the name of the preference to be created.
- **object_name**: Specify the name of the preference type.

**See Also:** For a complete list of preference types and their associated attributes, see Chapter 2, "Oracle Text Indexing Elements".

Examples

**Creating Text-only Index**

The following example creates a lexer preference that specifies a text-only index. It does so by creating a BASIC_LEXER preference called `my_lexer` with `CTX_DDL.CREATE_PREFERENCE`. It then calls `CTX_DDL.SET_ATTRIBUTE` twice, first specifying YES for the `INDEX_TEXT` attribute, then specifying NO for the `INDEX_THEMES` attribute.

```
begin
  ctx_ddl.create_preference('my_lexer', 'BASIC_LEXER');
  ctx_ddl.set_attribute('my_lexer', 'INDEX_TEXT', 'YES');
  ctx_ddl.set_attribute('my_lexer', 'INDEX_THEMES', 'NO');
end;
```

**Specifying File Data Storage**

The following example creates a data storage preference called `mypref` that tells the system that the files to be indexed are stored in the operating system. The example then uses `CTX_DDL.SET_ATTRIBUTE` to set the `PATH` attribute of to the directory `/docs`. 
begin
ctx_ddl.create_preference('mypref', 'FILE_DATASTORE');
ctx_ddl.set_attribute('mypref', 'PATH', '/docs');
end;

**See Also:** For more information about data storage, see "Datastore Types" in Chapter 2, "Oracle Text Indexing Elements".

### Creating Master/Detail Relationship

You can use CTX_DDL.CREATE_PREFERENCE to create a preference with DETAIL_DATASTORE. You use CTX_DDL.SET_ATTRIBUTE to set the attributes for this preference. The following example shows how this is done:

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

**See Also:** For more information about master/detail, see "DETAIL_DATASTORE" in Chapter 2, "Oracle Text Indexing Elements".

### 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;
Creating Preferences with No Attributes
When you create preferences with types that have no attributes, you need only create the preference, as in the following example which sets the filter to the NULL_FILTER:

begin
    ctx_ddl.create_preference('my_null_filter', 'NULL_FILTER');
end;

Related Topics

SET_ATTRIBUTE
DROP_PREFERENCE
CREATE INDEX in Chapter 1, "SQL Statements and Operators".
ALTER INDEX in Chapter 1, "SQL Statements and Operators".
Chapter 2, "Oracle Text Indexing Elements"
CREATE_SECTION_GROUP

Creates a section group for defining sections in a text column.

When you create a section group, you can add to it zone, field, or special sections with ADD_ZONE_SECTION, ADD_FIELD_SECTION, ADD_MDATA_SECTION, or ADD_SPECIAL_SECTION.

When you index, 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:

<table>
<thead>
<tr>
<th>Section Group Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL_SECTION_GROUP</td>
<td>Use this group type when you define no sections or when you define only SENTENCE or PARAGRAPH sections. This is the default.</td>
</tr>
</tbody>
</table>
| BASIC_SECTION_GROUP            | Use this group type for defining sections where the start and end tags are of the form `<A>` and `</A>`.
|                                | Note: This group type does not support input such as unbalanced parentheses, comments tags, and attributes. Use HTML_SECTION_GROUP for this type of input. |
| HTML_SECTION_GROUP             | Use this group type for indexing HTML documents and for defining sections in HTML documents. |
The following command creates a section group called `htmgrou` with the HTML group type.

```sql
begin
  ctx_ddl.create_section_group('htmgrou', 'HTML_SECTION_GROUP');
end;
```

---

## Section Group Preference

<table>
<thead>
<tr>
<th>Section Group Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML_SECTION_GROUP</td>
<td>Use this group type for indexing XML documents and for defining sections in XML documents.</td>
</tr>
</tbody>
</table>
| AUTO_SECTION_GROUP       | Use this group type to automatically create a zone section for each start-tag/end-tag pair in an XML document. The section names derived from XML tags are case sensitive as in XML. Attribute sections are created automatically for XML tags that have attributes. Attribute sections are named in the form attribute@tag. Stop sections, empty tags, processing instructions, and comments are not indexed. The following limitations apply to automatic section groups:  
  - You cannot add zone, field, or special sections to an automatic section group.  
  - Automatic sectioning does not index XML document types (root elements.) However, you can define stop sections with document type.  
  - The length of the indexed tags, including prefix and namespace, cannot exceed 64 characters. Tags longer than this are not indexed. |
| PATH_SECTION_GROUP       | Use this group type to index XML documents. Behaves like the AUTO_SECTION_GROUP. The difference is that with this section group you can do path searching with the INPATH and HASPATH operators. Queries are also case-sensitive for tag and attribute names. |
| NEWS_SECTION_GROUP       | Use this group for defining sections in newsgroup formatted documents according to RFC 1036. |

---

**Example**

The following command creates a section group called `htmgrou` with the HTML group type.

```sql
begin
  ctx_ddl.create_section_group('htmgrou', '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.

```sql
begin
ctx_ddl.create_section_group('auto', 'AUTO SECTION_GROUP');
end;
```

**Related Topics**

- **WITHIN** operator in Chapter 3, "CONTAINS Query Operators".
- "Section Group Types" in Chapter 2, "Oracle Text Indexing Elements".
- **ADD ZONE SECTION**
- **ADD FIELD SECTION**
- **ADD MDATA SECTION**
- **ADD SPECIAL SECTION**
- **REMOVE SECTION**
- **DROP SECTION_GROUP**
Use this procedure to create a new, empty stoplist. Stoplists can contain words or themes that are not to be indexed.

You can also create multi-language stoplists to hold language-specific stopwords. A multi-language stoplist is useful when you index a table that contains documents in different languages, such as English, German, and Japanese. When you do so, your text table must contain a language column.

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

You can specify a stoplist in the parameter string of CREATE INDEX or ALTER INDEX to override the default stoplist CTXSYS.DEFAULT_STOPLIST.

Syntax

```
CTX_DDL.CREATE_STOPLIST(
    stoplist_name IN VARCHAR2,
    stoplist_type IN VARCHAR2 DEFAULT 'BASIC_STOPLIST');
```

**stoplist_name**
Specify the name of the stoplist to be created.

**stoplist_type**
Specify BASIC_STOPLIST to create a stoplist for a single language. This is the default.

Specify MULTI_STOPLIST to create a stoplist with language-specific stopwords.

At indexing time, the language column of each document is examined, and only the stopwords for that language are eliminated. At query time, the session language setting determines the active stopwords, like it determines the active lexer when using the multi-lexer.

**Note:** When indexing a multi-language table with a multi-language stoplist, your table must have a language column.
Example

**Single Language Stoplist**
The following code creates a stoplist called `mystop`:

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

**Multi-Language Stoplist**
The following code creates a multi-language stoplist called `multistop` and then adds two language-specific stopwords:

```
begin
  ctx_ddl.create_stoplist('multistop', 'MULTI_STOPLIST');
  ctx_ddl.add_stopword('mystop', 'Die', 'german');
  ctx_ddl.add_stopword('mystop', 'Or', 'english');
end;
```

**Related Topics**

- ADD_STOPWORD
- ADD_STOPCLASS
- ADD_STOPTHEME
- DROP_STOPLIST

CREATE INDEX in Chapter 1, "SQL Statements and Operators".
ALTER INDEX in Chapter 1, "SQL Statements and Operators".
Appendix E, "Supplied Stoplists"
DROP_INDEX_SET

Drops a CTXCAT index set created with CTX_DDL.CREATE_INDEX_SET.

Syntax

CTX_DDL.DROP_INDEX_SET(set_name in varchar2);

set_name
Specify the name of the index set to drop.
Dropping an index set drops all of the sub-indexes it contains.
Drops a policy created with CTX_DDL.CREATE_POLICY.

Syntax

CTX_DDL.DROP_POLICY(policy_name IN VARCHAR2);

policy_name
Specify the name of the policy to drop.
DROP_PREFERENCE

The DROP_PREFERENCE procedure deletes the specified preference from the Text data dictionary. Dropping a preference does not affect indexes that have already been created using that preference.

Syntax

```
CTX_DDL.DROP_PREFERENCE(preference_name IN VARCHAR2);
```

**preference_name**
Specify the name of the preference to be dropped.

Example

The following code drops the preference my_lexer.

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

Related Topics

See also CTX_DDL.CREATE_PREFERENCE.
The DRP_SECION_GROUP procedure deletes the specified section group, as well as all the sections in the group, from the Text data dictionary.

**Syntax**

```sql
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:

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

**Related Topics**

See also CTX_DDL.CREATE_SECTION_GROUP.
Drops a stoplist from the Text data dictionary. When you drop a stoplist, you must re-create or rebuild the index for the change to take effect.

Syntax

```sql
CTX_DDL.DROP_STOPLIST(stoplist_name in varchar2);
```

**stoplist_name**
Specify the name of the stoplist.

Example

The following code drops the stoplist `mystop`:

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

Related Topics

See also `CTX_DDL.CREATE_STOPLIST`. 
Use this procedure to optimize the index. You optimize your index after you synchronize it. Optimizing an index removes old data and minimizes index fragmentation, which can improve query response time. Querying and DDL may proceed while optimization takes place.

You can optimize in fast, full, rebuild, token, or token-type mode.

- Fast mode compacts data but does not remove rows.
- Full mode compacts data and removes rows.
- Optimize in rebuild mode rebuilds the $I$ table (the inverted list table) in its entirety. Rebuilding an index is often significantly faster than performing a full optimization, and is more likely to result in smaller indexes, especially if the index is heavily fragmented.

Rebuild optimization creates a more compact copy of the $I$ table, and then switches the original $I$ table and the copy. The rebuild operation will therefore require enough space to store the copy as well as the original. (If redo logging is enabled, then additional space is required in the redo log as well). At the end of the rebuild operation, the original $I$ table is dropped, and the space can be reused.

- In token mode, you specify a specific token to be optimized (for example, all rows with documents containing the word elections). You can use this mode to optimize index tokens that are frequently searched, without spending time on optimizing tokens that are rarely referenced. An optimized token can improve query response time (but only for queries on that token).

- Token-type optimization is similar to token mode, except that the optimization is performed on field sections or MDATA sections (for example, sections with an <A> tag). This is useful in keeping critical field or MDATA sections optimal.

A common strategy for optimizing indexes is to perform regular token optimizations on frequently referenced terms, and to perform rebuild optimizations less frequently. (Use CTX_REPORT.QUERY_LOG_SUMMARY to find out which queries are made most frequently.) You can perform full, fast, or token-type optimizations instead of token optimizations.

Some users choose to perform frequent time-limited full optimizations along with occasional rebuild optimizations.
Using this procedure to optimize your index is recommended over using the `ALTER INDEX` statement.

Optimization of a large index may take a long time. To monitor the progress of a lengthy optimization, log the optimization with `CTX_OUTPUT.START_LOG` and check the resultant log file from time to time.

### Syntax

```sql
CTX_DDL.OPTIMIZE_INDEX(
 idx_name   IN  VARCHAR2,
 optlevel   IN  VARCHAR2,
 maxtime    IN  NUMBER DEFAULT NULL,
 token      IN VARCHAR2 DEFAULT NULL,
 part_name  IN VARCHAR2 DEFAULT NULL,
 token_type IN NUMBER DEFAULT NULL,
 parallel_degree IN VARCHAR2);
);
```

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAST or CTX_DDL.OPTLEVEL_FAST</td>
<td>This method compacts fragmented rows. However, old data is not removed.</td>
</tr>
<tr>
<td>FULL or CTX_DDL.OPTLEVEL_FULL</td>
<td>In this mode you can optimize the entire index or a portion of the index. This method compacts rows and removes old data (deleted rows). Optimizing in full mode runs even when there are no deleted rows.</td>
</tr>
</tbody>
</table>
maxtime
Specify maximum optimization time, in minutes, for FULL optimize.

When you specify the symbol CTX_DDL.MAXTIME.UNLIMITED (or pass in NULL), the entire index is optimized. This is the default.

token
Specify the token to be optimized.

part_name
If your index is a local index, you must specify the name of the index partition to synchronize otherwise an error is returned.

If your index is a global index, specify NULL, which is the default.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REBUILD or CTX_DDL.OPTLEVEL_REBUILD</td>
<td>This optlevel rebuilds the $I$ table (the inverted list table) to produce more compact token info rows. Like FULL optimize, this mode also deletes information pertaining to deleted rows of the base table. REBUILD is not supported for CTXCAT, CTXRULE, or CTXXPATH indexes. REBUILD optimization is also not supported for CONTEXT indexes that have substring indexing enabled. REBUILD is not supported when the $I$ table is partitioned. PARALLEL REBUILD optimization is permitted.</td>
</tr>
<tr>
<td>TOKEN or CTX_DDL.OPTLEVEL_TOKEN</td>
<td>This method lets you specify a specific token to be optimized. Oracle Text does a FULL optimization on the token you specify with token. Use this method to optimize those tokens that are searched frequently. Token optimization is not supported for CTXRULE indexes.</td>
</tr>
<tr>
<td>TOKEN_TYPE or CTX_DDL.OPTLEVEL_TOKEN_TYPE</td>
<td>This optlevel optimizes on demand all tokens in the index matching the input token type. When optlevel is TOKEN_TYPE, token_type must be provided. TOKEN_TYPE performs FULL optimize on any token of the input token_type. Like a TOKEN optimize, TOKEN_TYPE optimize does not change the FULL optimize state, and runs to completion on each invocation.</td>
</tr>
</tbody>
</table>
token_type
Specify the token_type to be optimized.

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

Examples
The following two examples are equivalent ways of optimizing an index using fast optimization:

begin
  ctx_ddl.optimize_index('myidx','FAST');
end;

begin
  ctx_ddl.optimize_index('myidx',CTX_DDL.OPTLEVEL_FAST);
end;

The following example optimizes the index token Oracle:

begin
  ctx_ddl.optimize_index('myidx','token', TOKEN=>'Oracle');
end;

To optimize all tokens of field section MYSEC in index MYINDEX:

begin
  ctx_ddl.optimize_index('myindex', ctx_ddl.optlevel_token_type,
    token_type=> ctx_report.token_type('myindex','field mysec text'));
end;

Related Topics
See also CTX_DDL.SYNC_INDEX and ALTER INDEX in Chapter 1, "SQL Statements and Operators".
Removes the index with the specified column list from a CTXCAT index set preference.

**Note:** This procedure does not remove a CTXCAT sub-index from the existing index. To do so, you must drop your index and re-index with the modified index set preference.

**Syntax**

```sql
CTX_DDL.REMOVE_INDEX(
    set_name in varchar2,
    column_list in varchar2
    language in varchar2 default NULL
);
```

- **set_name**
  Specify the name of the index set

- **column_list**
  Specify the name of the column list to remove.
REMOVE_MDATA

Use this procedure to remove metadata values, which are associated with an MDATA section, from a document. Only the owner of the index is allowed to call ADD_MDATA and REMOVE_MDATA.

Syntax

```sql
CTX_DDL.REMOVE_MDATA(
    idx_name           IN VARCHAR2,
    section_name       IN VARCHAR2,
    values             SYS.ODCIVARCHAR2LIST,
    rowids             SYS.ODCIRIDLIST,
    [part_name]        IN VARCHAR2
);
```

**idx_name**
Name of the text index that contains the named rowids.

**section_name**
Name of the MDATA section.

**values**
List of metadata values. If a metadata value contains a comma, the comma must be escaped with a backslash.

**rowids**
rowids from which to remove the metadata values.

**[part_name]**
Name of the index partition, if any. Must be provided for local partitioned indexes and must be NULL for global indexes.

Example

This example removes the MDATA value blue from the MDATA section BGCOLOR.

```sql
ctx_ddl.remove_mdata('idx_docs', 'bgcolor', 'blue', 'rows');
```
Related Topics

See also "ADD_MDATA" on page 7-11; "ADD_MDATA_SECTION" on page 7-14; "MDATA" on page 3-27; as well as the Section Searching chapter of the Oracle Text Application Developer's Guide.
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:

```sql
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:

```sql
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:

```sql
begin
    ctx_ddl.remove_section('htmgroupl', 'Title');
end;
```
Related Topics

ADD_FIELD_SECTION
ADD_SPECIAL_SECTION
ADD_ZONE_SECTION
REMOVES 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

```sql
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**:

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

Related Topics

ADD_STOPTHEME
Removes a stopword from a stoplist. To have the removal of a stopword be reflected in the index, you must rebuild your index.

Syntax

```sql
CTX_DDL.REMOVE_STOPWORD(
  stoplist_name  in   varchar2,
  stopword       in   varchar2,
  language       in   varchar2 default NULL
);
```

- `stoplist_name`: Specify the name of the stoplist.
- `stopword`: Specify the stopword to be removed from `stoplist_name`.
- `language`: Specify the language of `stopword` to remove when the stoplist you specify with `stoplist_name` is of type `MULTI_STOPLIST`. You must specify the Globalization Support name or abbreviation of an Oracle Text-supported language. You can also remove ALL stopwords.

Example

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

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

Related Topics

- `ADD_STOPWORD`
Use this procedure to replace metadata in local domain indexes at the global level.

**Syntax**

```sql
CTX_DDL.REPLACE_INDEX_METADATA(idx_name IN VARCHAR2,
                                parameter_string IN VARCHAR2);
```

- **idx_name**
  Specify the name of the index whose metadata you want to replace.

- **parameter_string**
  Specify the parameter string to be passed to `ALTER INDEX`. This must begin with 'REPLACE_METADATA'.

**Notes**

`ALTER INDEX REBUILD PARAMETER ('REPLACE_METADATA')` does not work for a local partitioned index at the index (global) level; you cannot, for example, use that `ALTER INDEX` syntax to change a global preference, such as filter or lexer type, without rebuilding the index. Therefore, `CTX_DDL.REPLACE_INDEX_METADATA` is provided as a method of overcoming this limitation of `ALTER INDEX`.

Though it is meant as a way to replace metadata for a local partitioned index, `CTX_DDL.REPLACE_INDEX_METADATA` can be used on a global index, as well.

`REPLACE_INDEX_METADATA` cannot be used to change the sync type at the partition level; that is, `parameter_string` cannot be 'REPLACE_METADATA_SYNC'. For that purpose, use `ALTER INDEX REBUILD PARTITION` to change the sync type at the partition level.

**Related Topics**

"ALTER INDEX REBUILD Syntax" on page 1-3
Sets a preference attribute. You use this procedure after you have created a preference with CTX_DDL.CREATE_PREFERENCE.

**Syntax**

```sql
CTX_DDL.SET_ATTRIBUTE(preference_name IN VARCHAR2,
                      attribute_name  IN VARCHAR2,
                      attribute_value IN VARCHAR2);
```

- **preference_name**
  Specify the name of the preference.

- **attribute_name**
  Specify the name of the attribute.

- **attribute_value**
  Specify the attribute value. You can specify boolean values as TRUE or FALSE, T or F, YES or NO, Y or N, ON or OFF, or 1 or 0.

**Example**

**Specifying File Data Storage**

The following example creates a data storage preference called `filepref` that tells the system that the files to be indexed are stored in the operating system. The example then uses CTX_DDL.SET_ATTRIBUTE to set the `PATH` attribute to the directory `/docs`.

```sql
begin
  ctx_ddl.create_preference('filepref', 'FILE_DATASTORE');
  ctx_ddl.set_attribute('filepref', 'PATH', '/docs');
end;
```

**See Also:** For more information about data storage, see “Datastore Types” in Chapter 2, “Oracle Text Indexing Elements”.

For more examples of using SET_ATTRIBUTE, see CREATE_PREFERENCE.
Synchronizes the index to process inserts, updates, and deletes to the base table.

**Syntax**

```sql
CTX_DDL.SYNC_INDEX(
  idx_name    IN  VARCHAR2 DEFAULT NULL,
  memory      IN VARCHAR2 DEFAULT NULL,
  part_name   IN VARCHAR2 DEFAULT NULL,
  parallel_degree IN NUMBER DEFAULT 1);
```

**idx_name**
Specify the name of the index.

**memory**
Specify the runtime memory to use for synchronization. This value overrides the `DEFAULT_INDEX_MEMORY` system parameter.

The memory parameter specifies the amount of memory Oracle Text uses for the synchronization operation before flushing the index to disk. Specifying a large amount of memory:

- improves indexing performance because there is less I/O
- improves query performance and maintenance because there is less fragmentation

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

**part_name**
If your index is a local index, you must specify the name of the index partition to synchronize otherwise an error is returned.

If your index is a global index, specify NULL, which is the default.

**parallel_degree**
Specify the degree to run parallel synchronize. A number greater than 1 turns on parallel synchronize. The actual degree of parallelism might be smaller depending on your resources.
Example

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

begin
ctx_ddl.sync_index('myindex', '2M');
end;

The following example synchronizes the part1 index partition with 2 megabytes of memory:

begin
ctx_ddl.sync_index('myindex', '2M', 'part1');
end;

Related Topics

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

Removes a set attribute from a preference.

Syntax

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

**preference_name**
Specify the name of the preference.

**attribute_name**
Specify the name of the attribute.

Example

**Enabling/Disabling Alternate Spelling**

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

```sql
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:

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

Related Topics

- SET_ATTRIBUTE on page 7-61
UPDATE_POLICY

Updates a policy created with CREATE_POLICY. Replaces the preferences of the policy. Null arguments are not replaced.

Syntax

CTX_DDL.UPDATE_POLICY(
    policy_name  IN VARCHAR2 DEFAULT NULL,
    filter       IN VARCHAR2 DEFAULT NULL,
    section_group IN VARCHAR2 DEFAULT NULL,
    lexer        IN VARCHAR2 DEFAULT NULL,
    stoplist     IN VARCHAR2 DEFAULT NULL,
    wordlist     IN VARCHAR2 DEFAULT NULL);

policy_name
Specify the name of the policy to update.

filter
Specify the filter preference to use.

section_group
Specify the section group to use.

lexer
Specify the lexer preference to use.

stoplist
Specify the stoplist to use.

wordlist
Specify the wordlist to use.
This chapter describes the CTX_DOC PL/SQL package for requesting document services, such as highlighting extracted text or generating a list of themes for a document.

Many of these procedures exist in two versions: those that make use of indexes, and those that don't. Those that don't are called "policy-based" procedures. They are offered because there are times when you might like to use document services on a single document without creating a context index in advance. Policy-based procedures enable you to do this.

The policy_* procedures mirror the conventional in-memory document services and are used with policy name replacing index name, and document of type VARCHAR2, CLOB, BLOB or BFILE replacing textkey. Thus, you need not create an index to obtain document services output with these procedures.

The CTX_DOC package includes the following procedures and functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILTER</td>
<td>Generates a plain text or HTML version of a document</td>
</tr>
<tr>
<td>GIST</td>
<td>Generates a Gist or theme summaries for a document</td>
</tr>
<tr>
<td>HIGHLIGHT</td>
<td>Generates plain text or HTML highlighting offset information for a document</td>
</tr>
<tr>
<td>IFILTER</td>
<td>Generates a plain text version of binary data. Can be called from a USER_DATASTORE procedure.</td>
</tr>
<tr>
<td>MARKUP</td>
<td>Generates a plain text or HTML version of a document with query terms highlighted</td>
</tr>
<tr>
<td>PKENCODE</td>
<td>Encodes a composite textkey string (value) for use in other CTX_DOC procedures</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>POLICY_FILTER</td>
<td>Generates a plain text or HTML version of a document, without requiring an index.</td>
</tr>
<tr>
<td>POLICY_GIST</td>
<td>Generates a Gist or theme summaries for a document, without requiring an index.</td>
</tr>
<tr>
<td>POLICY_HIGHLIGHT</td>
<td>Generates plain text or HTML highlighting offset information for a document, without requiring an index.</td>
</tr>
<tr>
<td>POLICY_MARKUP</td>
<td>Generates a plain text or HTML version of a document with query terms highlighted, without requiring an index.</td>
</tr>
<tr>
<td>POLICY_THEMES</td>
<td>Generates a list of themes for a document, without requiring an index.</td>
</tr>
<tr>
<td>POLICY_TOKENS</td>
<td>Generates all index tokens for a document, without requiring an index.</td>
</tr>
<tr>
<td>SET_KEY_TYPE</td>
<td>Sets CTX_DOC procedures to accept rowid or primary key document identifiers.</td>
</tr>
<tr>
<td>THEMES</td>
<td>Generates a list of themes for a document</td>
</tr>
<tr>
<td>TOKENS</td>
<td>Generates all index tokens for a document</td>
</tr>
</tbody>
</table>
FILTER

Use the CTX_DOC.FILTER procedure to generate either a plain text or HTML version of a document. You can store the rendered document in either a result table or in memory. This procedure is generally called after a query, from which you identify the document to be filtered.

Note: The resultant HTML document does not include graphics.

Syntax 1: In-memory Result Storage

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

Syntax 2: Result Table Storage

```sql
CTX_DOC.FILTER(
    index_name  IN VARCHAR2,
    textkey     IN VARCHAR2,
    restab      IN VARCHAR2,
    query_id    IN NUMBER DEFAULT 0,
    plaintext   IN BOOLEAN  DEFAULT FALSE);
```

**index_name**
Specify the name of the index associated with the text column containing the document identified by textkey.

**textkey**
Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can be one of the following:

- a single column primary key value
- encoded specification for a composite (multiple column) primary key. Use CTX_DOC.PKENCODE.
- the rowid of the row containing the document
You toggle between primary key and rowid identification using CTX_DOC.SET_KEY_TYPE.

**restab**

You can specify that this procedure store the marked-up text to either a table or to an in-memory CLOB.

To store results to a table specify the name of the table. The result table must exist before you make this call.

**See Also:** “Filter Table” in Appendix A, “Result Tables” for more information about the structure of the filter result table.

To store results in memory, specify the name of the CLOB locator. If restab is NULL, a temporary CLOB is allocated and returned. You must de-allocate the locator after using it with DBMS_LOB.FREETEMPORARY().

If restab is not NULL, the CLOB is truncated before the operation.

**query_id**

Specify an identifier to use to identify the row inserted into restab.

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

**plaintext**

Specify TRUE to generate a plaintext version of the document. Specify FALSE to generate an HTML version of the document if you are using the INSO filter or indexing HTML documents.

**Example**

**In-Memory Filter**

The following code shows how to filter a document to HTML in memory.

```sql
declare
    mklob clob;
    amt number := 40;
    line varchar2(80);
begin
    ctx_doc.filter('myindex','1', mklob, FALSE);
    -- mklob is NULL when passed-in, so ctx-doc.filter will allocate a temporary
    -- CLOB for us and place the results there.
end;
```

8-4 Oracle Text Reference
dbms_lob.read(mklob, amt, 1, line);
dbms_output.put_line('FIRST 40 CHAR 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;
Use the CTX_DOC.GIST procedure to generate gist and theme summaries for a document. You can generate paragraph-level or sentence-level gists or theme summaries.

Syntax 1: In-Memory Storage

```sql
CTX_DOC.GIST(
    index_name    IN VARCHAR2,
    textkey       IN VARCHAR2,
    restab        IN OUT CLOB,
    glevel        IN VARCHAR2 DEFAULT 'P',
    pov           IN VARCHAR2 DEFAULT 'GENERIC',
    numParagraphs IN NUMBER DEFAULT 16,
    maxPercent    IN NUMBER DEFAULT 10,
    num_themes   IN NUMBER DEFAULT 50);
```

Syntax 2: Result Table Storage

```sql
CTX_DOC.GIST(
    index_name    IN VARCHAR2,
    textkey       IN VARCHAR2,
    restab        IN VARCHAR2,
    query_id      IN NUMBER DEFAULT 0,
    glevel        IN VARCHAR2 DEFAULT 'P',
    pov           IN VARCHAR2 DEFAULT NULL,
    numParagraphs IN NUMBER DEFAULT 16,
    maxPercent    IN NUMBER DEFAULT 10,
    num_themes     IN NUMBER DEFAULT 50);
```

**index_name**
Specify the name of the index associated with the text column containing the document identified by `textkey`.

**textkey**
Specify the unique identifier (usually the primary key) for the document.

The `textkey` parameter can be one of the following:

- a single column primary key value
- an encoded specification for a composite (multiple column) primary key. To encode a composite `textkey`, use the `CTX_DOC.PKENCODE` procedure.
the rowid of the row containing the document

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

restart

You can specify that this procedure store the gist and theme summaries to either a table or to an in-memory CLOB.

To store results to a table specify the name of the table.

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

To store results in memory, specify the name of the CLOB locator. If restart is NULL, a temporary CLOB is allocated and returned. You must de-allocate the locator after using it.

If restart is not NULL, the CLOB is truncated before the operation.

query_id

Specify an identifier to use to identify the row(s) inserted into restart.

glevel

Specify the type of gist or theme summary to produce. The possible values are:

- P for paragraph
- S for sentence

The default is P.

pov

Specify whether a gist or a single theme summary is generated. The type of gist or theme summary generated (sentence-level or paragraph-level) depends on the value specified for glevel.

To generate a gist for the entire document, specify a value of 'GENERIC' for pov. To generate a theme summary for a single theme in a document, specify the theme as the value for pov.

When using result table storage and you do not specify a value for pov, this procedure returns the generic gist plus up to fifty theme summaries for the document.
When using in-memory result storage to a CLOB, you must specify a pov. However, if you do not specify pov, this procedure generates only a generic gist for the document.

**Note:** The pov parameter is case sensitive. To return a gist for a document, specify ‘GENERIC’ in all uppercase. To return a theme summary, specify the theme exactly as it is generated for the document.

Only the themes generated by THEMES for a document can be used as input for pov.

**numParagraphs**
Specify the maximum number of document paragraphs (or sentences) selected for the document gist or theme summaries. The default is 16.

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

This means that the system always returns the smallest size gist or theme summary.

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

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

This means that the system always returns the smallest size gist or theme summary.

**num_themes**
Specify the number of theme summaries to produce when you do not specify a value for pov. For example, if you specify 10, this procedure returns the top 10 theme summaries. The default is 50.
If you specify 0 or NULL, this procedure returns all themes in a document. If the document contains more than 50 themes, only the top 50 themes show conceptual hierarchy.

**Examples**

**In-Memory Gist**
The following example generates a non-default size generic gist of at most 10 paragraphs. The result is stored in memory in a CLOB locator. The code then de-allocates the returned CLOB locator after using it.

```sql
set serveroutput on;
declare
  gklob clob;
  amt number := 40;
  line varchar2(80);
begin
  ctx_doc.gist('newsindex','34',gklob, pov => 'GENERIC',numParagraphs => 10);
  -- gklob is NULL when passed-in, so ctx-doc.gist will allocate a temporary CLOB for us and place the results there.
  dbms_lob.read(gklob, amt, 1, line);
  dbms_output.put_line('FIRST 40 CHARs ARE:'||line);
  -- have to de-allocate the temp lob
  dbms_lob.freetemporary(gklob);
end;
```

**Result Table Gists**
The following example creates a gist table called CTX_GIST:

```sql
create table CTX_GIST (query_id number,
                        pov varchar2(80),
                        gist CLOB);
```

**Gists and Theme Summaries**
The following example returns a default sized paragraph level gist for document 34 as well as the top 10 theme summaries in the document:

```sql
begin
  ctx_doc.gist('newsindex','34','CTX_GIST', 1, num_themes=>10);
end;
```
The following example generates a non-default size gist of at most 10 paragraphs:
begin
  ctx_doc.gist('newsindex','34','CTX_GIST',1,pov =>'GENERIC',numParagraphs=>10);
end;

The following example generates a gist whose number of paragraphs is at most 10 percent of the total paragraphs in document:
begin
  ctx_doc.gist('newsindex','34','CTX_GIST',1,pov => 'GENERIC',  maxPercent => 10);
end;

**Theme Summary**
The following example returns a paragraph level theme summary for *insects* for document 34. The default theme summary size is returned.
begin
  ctx_doc.gist('newsindex','34','CTX_GIST',1, pov => 'insects');
end;
Use the CTX_DOC.HIGHLIGHT procedure to generate highlight offsets for a document. The offset information is generated for the terms in the document that satisfy the query you specify. These highlighted terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.

You can generate highlight offsets for either plaintext or HTML versions of the document. The table returned by CTX_DOC.HIGHLIGHT does not include any graphics found in the original document. You 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**

```sql
CTX_DOC.HIGHLIGHT(
    index_name  IN VARCHAR2,
    textkey     IN VARCHAR2,
    text_query  IN VARCHAR2,
    restab      IN OUT NOCOPY HIGHLIGHT_TAB,
    plaintext   IN BOOLEAN  DEFAULT FALSE);
```

**Syntax 2: Result Table Storage**

```sql
CTX_DOC.HIGHLIGHT(
    index_name  IN VARCHAR2,
    textkey     IN VARCHAR2,
    text_query  IN VARCHAR2,
    restab      IN VARCHAR2,
    query_id    IN NUMBER   DEFAULT 0,
    plaintext   IN BOOLEAN  DEFAULT FALSE);
```

**index_name**
Specify the name of the index associated with the text column containing the document identified by `textkey`.

**textkey**
Specify the unique identifier (usually the primary key) for the document.
The `textkey` parameter can be one of the following:

- a single column primary key value
- encoded specification for a composite (multiple column) primary key. Use the `CTX_DOC.PKENCODE` procedure.
- the rowid of the row containing the document

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

**text_query**

Specify the original query expression used to retrieve the document. If NULL, no highlights are generated.

If `text_query` includes wildcards, stemming, fuzzy matching which result in stopwords being returned, `HIGHLIGHT` does not highlight the stopwords.

If `text_query` contains the threshold operator, the operator is ignored. The `HIGHLIGHT` procedure always returns highlight information for the entire result set.

**restab**

You can specify that this procedure store highlight offsets to either a table or to an in-memory PL/SQL table.

To store results to a table specify the name of the table. The table must exist before you call this procedure.

**See Also:** see "Highlight Table" in Appendix A, "Result Tables" for more information about the structure of the highlight result table.

To store results to an in-memory table, specify the name of the in-memory table of type `CTX_DOC.HIGHLIGHT_TAB`. The `HIGHLIGHT_TAB` datatype is defined as follows:

```plsql
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 \texttt{restab}.

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

plaintext
Specify \texttt{TRUE} to generate a plaintext offsets of the document.

Specify \texttt{FALSE} to generate HTML offsets of the document if you are using the INSO filter or indexing HTML documents.

\section*{Examples}

\subsection*{Create Highlight Table}
Create the highlight table to store the highlight offset information:

```sql
create table hightab(query_id number,
    offset number,
    length number);
```

\subsection*{Word Highlight Offsets}
To obtain HTML highlight offset information for document 20 for the word \textit{dog}:

```sql
begin
    ctx_doc.highlight('newsindex', '20', 'dog', 'hightab', 0, FALSE);
end;
```

\subsection*{Theme Highlight Offsets}
Assuming the index \textit{newsindex} has a theme component, you obtain HTML highlight offset information for the theme query of \textit{politics} by issuing the following query:

```sql
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 \textit{politics} in the document.
Use this procedure when you need to filter binary data to text.

This procedure takes binary data (BLOB IN), filters the data through with the Inso filter, and writes the text version to a CLOB. (Any graphics in the original document are ignored.) CTX_DOC.IFILTER employs the safe callout, and it does not require an index to use, as CTX_DOC.FILTER does.

**Note:** This procedure will not be supported in future releases. Programs should make use of CTX_DOC.POLICY_FILTER instead.

**Requirements**

Because CTX_DOC.IFILTER employs the safe callout mechanism, the SQL*Net listener must be running and configured for extproc agent startup.

**Syntax**

```sql
CTX_DOC.IFILTER(data IN BLOB, text IN OUT NOCOPY CLOB);
```

- **data**
  Specify the binary data to be filtered.

- **text**
  Specify the destination CLOB. The filtered data is placed in here. This parameter must be a valid CLOB locator that is writable. Passing NULL or a non-writable CLOB will result in an error. Filtered text will be appended to the end of existing content, if any.

**Example**

The document text used in a MATCHES query can be VARCHAR2 or CLOB. It does not accept BLOB input, so you cannot match filtered documents directly. Instead, you must filter the binary content to CLOB using the INSO filter. Assuming the document data is in bind variable :doc_blob:

```sql
declare
doc_text clob;
begin
  -- create a temporary CLOB to hold the document text
```

---

8-14  Oracle Text Reference
IFILTER

doc_text := dbms_lob.createtemporary(doc_text, TRUE, DBMS_LOB.SESSION);

-- call ctx_doc.ifilter to filter the BLOB to CLOB data
ctx_doc.ifilter(:doc_blob, doc_text);

-- now do the matches query using the CLOB version
for c1 in (select * from queries where matches(query_string, doc_text)>0)
  loop
    -- do what you need to do here
    end loop;

  dbms_lob.freetemporary(doc_text);
end;
The CTX_DOC.MARKUP procedure takes a query specification and a document textkey and returns a version of the document in which the query terms are marked up. These marked-up terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.

You can set the marked-up output to be either plaintext or HTML. The marked-up document returned by CTX_DOC.MARKUP does not include any graphics found in the original document.

You can use one of the pre-defined 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.

**Note:** Oracle Text does not guarantee well-formed output from CTX_DOC.MARKUP, especially for terms that are already marked up with HTML or XML. In particular, unexpected nesting of markup tags may occasionally result.

### Syntax 1: In-Memory Result Storage

```
CTX_DOC.MARKUP (  
index_name IN VARCHAR2,  
textkey IN VARCHAR2,  
text_query IN VARCHAR2,  
restab IN OUT NOCOPY CLOB,  
plaintext IN BOOLEAN DEFAULT FALSE,  
tagset IN VARCHAR2 DEFAULT 'TEXT_DEFAULT',  
starttag IN VARCHAR2 DEFAULT NULL,  
endtag IN VARCHAR2 DEFAULT NULL,  
prevtag IN VARCHAR2 DEFAULT NULL,  
nexttag IN VARCHAR2 DEFAULT NULL);
```

### Syntax 2: Result Table Storage

```
CTX_DOC.MARKUP (  
index_name IN VARCHAR2,  
...)
```
textkey        IN VARCHAR2,
text_query     IN VARCHAR2,
restab         IN VARCHAR2,
query_id       IN NUMBER    DEFAULT 0,
plaintext      IN BOOLEAN   DEFAULT FALSE,
tagset         IN VARCHAR2  DEFAULT 'TEXT_DEFAULT',
starttag       IN VARCHAR2  DEFAULT NULL,
endtag         IN VARCHAR2  DEFAULT NULL,
prevtag        IN VARCHAR2  DEFAULT NULL,
nexttag        IN VARCHAR2  DEFAULT NULL);

index_name
Specify the name of the index associated with the text column containing the
document identified by textkey.

textkey
Specify the unique identifier (usually the primary key) for the document.
The textkey parameter can be one of the following:
- a single column primary key value
- encoded specification for a composite (multiple column) primary key. Use the
  CTX_DOC.PKENCODE procedure.
- the rowid of the row containing the document
You toggle between primary key and rowid identification using CTX_DOC.SET_ KEY_TYPE.

text_query
Specify the original query expression used to retrieve the document.
If text_query includes wildcards, stemming, fuzzy matching which result in
stopwords being returned, MARKUP does not highlight the stopwords.
If text_query contains the threshold operator, the operator is ignored. The MARKUP
procedure always returns highlight information for the entire result set.

restab
You can specify that this procedure store the marked-up text to either a table or to
an in-memory CLOB.
To store results to a table specify the name of the table. The result table must exist
before you call this procedure.
To store results in memory, specify the name of the CLOB locator. If `restab` is `NULL`, a temporary CLOB is allocated and returned. You must de-allocate the locator after using it.

If `restab` is not `NULL`, the CLOB is truncated before the operation.

**query_id**

Specify the identifier used to identify the row inserted into `restab`.

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

**plaintext**

Specify `TRUE` to generate plaintext marked-up document. Specify `FALSE` to generate a marked-up HTML version of document if you are using the INSO filter or indexing HTML documents.

**tagset**

Specify one of the following pre-defined tagsets. The second and third columns show how the four different tags are defined for each tagset:

<table>
<thead>
<tr>
<th>Tagset</th>
<th>Tag</th>
<th>Tag Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT_DEFAULT</td>
<td>starttag</td>
<td><code>&lt;&lt;&lt;</code></td>
</tr>
<tr>
<td></td>
<td>endtag</td>
<td><code>&gt;&gt;&gt;</code></td>
</tr>
<tr>
<td></td>
<td>prevtag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nexttag</td>
<td></td>
</tr>
<tr>
<td>HTML_DEFAULT</td>
<td>starttag</td>
<td><code>&lt;B&gt;</code></td>
</tr>
<tr>
<td></td>
<td>endtag</td>
<td><code>&lt;/B&gt;</code></td>
</tr>
<tr>
<td></td>
<td>prevtag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nexttag</td>
<td></td>
</tr>
<tr>
<td>HTML_NAVIGATE</td>
<td>starttag</td>
<td><code>&lt;A NAME=ctx%CURNUM&gt;&lt;B&gt;</code></td>
</tr>
<tr>
<td></td>
<td>endtag</td>
<td><code>&lt;/B&gt;&lt;/A&gt;</code></td>
</tr>
<tr>
<td></td>
<td>prevtag</td>
<td><code>&lt;A HREF=#ctx%PREVNUM&gt;&amp;lt;&lt;/A&gt;</code></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Offset Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%CURNUM</td>
<td>the current offset number</td>
</tr>
<tr>
<td>%PREVNUM</td>
<td>the previous offset number</td>
</tr>
<tr>
<td>%NEXTNUM</td>
<td>the next offset number</td>
</tr>
</tbody>
</table>

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.

```sql
set serveroutput on

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

begin
  ctx_doc.markup('myindex','1','dog & cat', mklob);
  -- mklob is NULL when passed-in, so ctx-doc.markup will allocate a temporary
  -- CLOB for us and place the results there.
  dbms_lob.read(mklob, amt, 1, line);
  dbms_output.put_line('FIRST 40 CHARS ARE:'||line);
  -- have to de-allocate the temp lob
  dbms_lob.freetemporary(mklob);
end;
```

**Markup Table**

Create the highlight markup table to store the marked-up document as follows:

```sql
create table markuptab (query_id  number,
                           document  clob);
```

**Word Highlighting in HTML**

You can also store your MARKUP results in a table. To create HTML highlight markup for the words `dog` or `cat` for document 23, issue the following statement:

```sql
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:

```plaintext
begin
  ctx_doc.markup(index_name => 'my_index',
                  textkey => '23',
                  text_query => 'about(politics)',
                  restab => 'markuptab',
                  query_id => '1',
                  tagset => 'HTML_DEFAULT');
end;
```
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**

```sql
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**

```sql
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.
Generates a plain text or an HTML version of a document. With this procedure, no CONTEXT index is required.

This procedure uses a trusted callout.

Syntax

```sql
ctx_doc.policy_filter(policy_name    in  VARCHAR2,
                      document       in [VARCHAR2|CLOB|BLOB|BFILE],
                      restab         in out nocopy CLOB,
                      plaintext      in BOOLEAN default FALSE);
```

**policy_name**
Specify the policy name created with CTX_DDL.CREATE_POLICY. Using an index name will result in an error.

**document**
Specify the document to filter.

**restab**
Specify the name of the result table.

**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.
POLICY_GIST

Generates a Gist or theme summary for document. You can generate paragraph-level or sentence-level gists or theme summaries. With this procedure, no CONTEXT index is required.

Syntax

ctx_doc.policy_gist(policy_name in VARCHAR2,
document in [VARCHAR2|CLOB|BLOB|BFILE],
restab in out nocopy CLOB,
glevel in VARCHAR2 default 'P',
pov in VARCHAR2 default 'GENERIC',
numParagraphs in VARCHAR2 default NULL,
maxPercent in NUMBER default NULL,
num_themes in NUMBER default 50);

policy_name
Specify the policy name created with CTX_DDL.CREATE_POLICY. Using an index name will result in an error.

document
Specify the document for which to generate the Gist or theme summary.

restab
Specify the name of the result table.

glevel
Specify the type of gist or theme summary to produce. The possible values are:

- P for paragraph
- S for sentence

The default is P.

pov
Specify whether a gist or a single theme summary is generated. The type of gist or theme summary generated (sentence-level or paragraph-level) depends on the value specified for glevel.
To generate a gist for the entire document, specify a value of 'GENERIC' for pov. To generate a theme summary for a single theme in a document, specify the theme as the value for pov.

When using result table storage and you do not specify a value for pov, this procedure returns the generic gist plus up to fifty theme summaries for the document.

**Note:** The pov parameter is case sensitive. To return a gist for a document, specify 'GENERIC' in all uppercase. To return a theme summary, specify the theme exactly as it is generated for the document.

Only the themes generated by THEMES for a document can be used as input for pov.

**numParagraphs**
Specify the maximum number of document paragraphs (or sentences) selected for the document gist or theme summaries. The default is 16.

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

This means that the system always returns the smallest size gist or theme summary.

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

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

This means that the system always returns the smallest size gist or theme summary.
num_themes
Specify the number of theme summaries to produce when you do not specify a value for pov. For example, if you specify 10, this procedure returns the top 10 theme summaries. The default is 50.

If you specify 0 or NULL, this procedure returns all themes in a document. If the document contains more than 50 themes, only the top 50 themes show conceptual hierarchy.
Generates plain text or HTML highlighting offset information for a document. With this procedure, no CONTEXT index is required.

The offset information is generated for the terms in the document that satisfy the query you specify. These highlighted terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.

You can generate highlight offsets for either plaintext or HTML versions of the document. You can apply the offset information to the same documents filtered with CTX_DOC.FILTER.

Syntax

```
ctx_doc.policy_highlight(policy_name in VARCHAR2,
document     in [VARCHAR2|CLOB|BLOB|BFILE],
text_query   in VARCHAR2,
restab       in out nocopy highlight_tab,
plaintext    in boolean FALSE);
```

**policy_name**
Specify the policy name created with CTX_DDL.CREATE_POLICY. Using an index name will result in an error.

**document**
Specify the document to generate highlighting offset information.

**text_query**
Specify the original query expression used to retrieve the document. If NULL, no highlights are generated.

If `text_query` includes wildcards, stemming, or fuzzy matching which result in stopwords being returned, this procedure does not highlight the stopwords.

If `text_query` contains the threshold operator, the operator is ignored. This procedure always returns highlight information for the entire result set.

**restab**
Specify the name of the result table. The table must exist before you call this procedure.
See Also: see "Highlight Table" in Appendix A, "Result Tables" for more information about the structure of the highlight result table.

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.
Generates plain text or HTML version of a document with query terms highlighted. With this procedure, no CONTEXT index is required.

The `CTX_DOC.POLICY_MARKUP` procedure takes a query specification and a document and returns a version of the document in which the query terms are marked up. These marked-up terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.

You can set the marked-up output to be either plaintext or HTML.

You can use one of the pre-defined tagsets for marking highlighted terms, including a tag sequence that enables HTML navigation.

**Syntax**

```sql
ctx_doc.policy_markup(policy_name in VARCHAR2,
document        in [VARCHAR2|CLOB|BLOB|BFILE],
text_query      in VARCHAR2,
restab          in out nocopy CLOB,
plaintext       in BOOLEAN default FALSE,
tagset          in VARCHAR2 default 'TEXT_DEFAULT',
starttag        in VARCHAR2 default NULL,
endtag          in VARCHAR2 default NULL,
prevtag         in VARCHAR2 default NULL,
nexttag         in VARCHAR2 default NULL);
```

**policy_name**

Specify the policy name created with `CTX_DDL.CREATE_POLICY`. Using an index name will result in an error.

**document**

Specify the document to generate highlighting offset information.

**text_query**

Specify the original query expression used to retrieve the document. If NULL, no highlights are generated.

If `text_query` includes wildcards, stemming, or fuzzy matching which result in stopwords being returned, this procedure does not highlight the stopwords.

If `text_query` contains the threshold operator, the operator is ignored. This procedure always returns highlight information for the entire result set.
**restab**

Specify the name of the result table. The table must exist before you call this procedure.

**See Also:** see "Markup Table" in Appendix A, "Result Tables" for more information about the structure of the highlight result table.

**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:

<table>
<thead>
<tr>
<th>Tagset</th>
<th>Tag</th>
<th>Tag Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT_DEFAULT</td>
<td>starttag</td>
<td><code>&lt;&lt;&lt;</code></td>
</tr>
<tr>
<td></td>
<td>endtag</td>
<td><code>&gt;&gt;&gt;</code></td>
</tr>
<tr>
<td></td>
<td>prevtag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nexttag</td>
<td></td>
</tr>
<tr>
<td>HTML_DEFAULT</td>
<td>starttag</td>
<td><code>&lt;B&gt;</code></td>
</tr>
<tr>
<td></td>
<td>endtag</td>
<td><code>&lt;/B&gt;</code></td>
</tr>
<tr>
<td></td>
<td>prevtag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nexttag</td>
<td></td>
</tr>
<tr>
<td>HTML_NAVIGATE</td>
<td>starttag</td>
<td><code>&lt;A NAME=ctx%CURNUM&gt;&lt;B&gt;</code></td>
</tr>
<tr>
<td></td>
<td>endtag</td>
<td><code>&lt;/B&gt;&lt;/A&gt;</code></td>
</tr>
<tr>
<td></td>
<td>prevtag</td>
<td><code>&lt;A HREF=#ctx%PREVNUM&gt;&amp;lt;&lt;/A&gt;</code></td>
</tr>
<tr>
<td></td>
<td>nexttag</td>
<td><code>&lt;A HREF=#ctx%NEXTNUM&gt;&amp;gt;&lt;/A&gt;</code></td>
</tr>
</tbody>
</table>

**starttag**

Specify the character(s) inserted by `MARKUP` to indicate the start of a highlighted term.

The sequence of `starttag`, `endtag`, `prevtag` and `nexttag` with regard to the highlighted word is as follows:
... prevtag starttag word endtag nexttag...

**endtag**
Specify the character(s) inserted by `MARKUP` to indicate the end of a highlighted term.

**prevtag**
Specify the markup sequence that defines the tag that navigates the user to the previous highlight.

In the markup sequences `prevtag` and `nexttag`, you can specify the following offset variables which are set dynamically:

<table>
<thead>
<tr>
<th>Offset Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%CURNUM</td>
<td>the current offset number</td>
</tr>
<tr>
<td>%PREVNUM</td>
<td>the previous offset number</td>
</tr>
<tr>
<td>%NEXTNUM</td>
<td>the next offset number</td>
</tr>
</tbody>
</table>

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.
POLICY_THEMES

Generates a list of themes for a document. With this procedure, no CONTEXT index is required.

Syntax

```sql
ctx_doc.policy_themes(policy_name in VARCHAR2,
                      document   in [VARCHAR2,CLOB,BLOB,BFILE],
                      restab      in out nocopy theme_tab,
                      full_themes in BOOLEAN default FALSE,
                      num_themes  in number    default 50);```

**policy_name**
Specify the policy you create with CTX_DDL.CREATE_POLICY. Using an index name will result in an error.

**document**
Specify the document for which to generate a list of themes.

**restab**
Specify the name of the result table.

**full_themes**
Specify whether this procedure generates a single theme or a hierarchical list of parent themes (full themes) for each document theme.

Specify **TRUE** for this procedure to write full themes to the THEME column of the result table.

Specify **FALSE** for this procedure to write single theme information to the THEME column of the result table. This is the default.

**num_themes**
Specify the maximum number of themes to retrieve. For example, if you specify 10, up to first 10 themes are returned for the document. The default is 50.

See Also: "Theme Table" in Appendix A, "Result Tables" for more information about the structure of the theme result table.
If you specify 0 or NULL, this procedure returns all themes in a document. If the document contains more than 50 themes, only the first 50 themes show conceptual hierarchy.

Example

Create a policy:

exec ctx_ddl.create_policy('mypolicy');

Run themes:

declare
   la      varchar2(200);
   rtab    ctx_doc.theme_tab;
begin
   ctx_doc.policy_themes('mypolicy',
      'To define true madness, What is''t but to be nothing but mad?', rtab);
   for i in 1..rtab.count loop
      dbms_output.put_line(rtab(i).theme||':'||rtab(i).weight);
   end loop;
end;
POLICY_TOKENS

Generate all index tokens for document. With this procedure, no CONTEXT index is required.

Syntax

```sql
ctx_doc.policy_tokens(policy_name in VARCHAR2,
                      document    in [VARCHAR2|CLOB|BLOB|BFILE],
                      restab       in out nocopy token_tab);
```

**policy_name**
Specify the policy name created with CTX_DDL.CREATE_POLICY. Using an index name will result in an error.

**document**
Specify the document for which to generate tokens.

**restab**
Specify the name of the result table.

The tokens returned are those tokens which are inserted into the index for the document. Stop words are not returned. Section tags are not returned because they are not text tokens.

Token tables can be named anything, but must include the following columns, with names and data types as follows.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_ID</td>
<td>NUMBER</td>
<td>The identifier for the results generated by a particular call to CTX_DOC.TOKENS (only populated when table is used to store results from multiple TOKEN calls)</td>
</tr>
<tr>
<td>TOKEN</td>
<td>VARCHAR2 (64)</td>
<td>The token string in the text.</td>
</tr>
<tr>
<td>OFFSET</td>
<td>NUMBER</td>
<td>The position of the token in the document, relative to the start of document which has a position of 1.</td>
</tr>
<tr>
<td>LENGTH</td>
<td>NUMBER</td>
<td>The character length of the token.</td>
</tr>
</tbody>
</table>
Example

Get tokens:

declare
  la     varchar2(200);
  rtab   ctx_doc.token_tab;
begin
  ctx_doc.policy_tokens('mypolicy',
    'To define true madness, What isn't but to be nothing but mad?',
    rtab);
  for i in 1..rtab.count loop
    dbms_output.put_line(rtab(i).offset||':'||rtab(i).token);
  end loop;
end;
SET_KEY_TYPE

Use this procedure to set the CTX_DOC procedures to accept either the ROWID or the PRIMARY_KEY document identifiers. This setting affects the invoking session only.

Syntax

    ctx_doc.set_key_type(key_type in varchar2);

key_type

Specify either ROWID or PRIMARY_KEY as the input key type (document identifier) for CTX_DOC procedures.

This parameter defaults to the value of the CTX_DOC_KEY_TYPE system parameter.

Note: When your base table has no primary key, setting key_type to PRIMARY_KEY is ignored. The textkey parameter you specify for any CTX_DOC procedure is interpreted as a ROWID.

Example

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

    begin
    ctx_doc.set_key_type('PRIMARY_KEY');
    ctx_doc.set_key_type('ROWID');
    end
Use the `CTX_DOC.THEMES` procedure to generate a list of themes for a document. You can store each theme as a row in either a result table or an in-memory PL/SQL table you specify.

**Syntax 1: In-Memory Table Storage**

```sql
CTX_DOC.THEMES(
    index_name      IN VARCHAR2,
    textkey         IN VARCHAR2,
    restab          IN OUT NOCOPY THEME_TAB,
    full_themes     IN BOOLEAN DEFAULT FALSE,
    num_themes       IN NUMBER DEFAULT 50);
```

**Syntax 2: Result Table Storage**

```sql
CTX_DOC.THEMES(
    index_name      IN VARCHAR2,
    textkey         IN VARCHAR2,
    restab          IN VARCHAR2,
    query_id        IN NUMBER DEFAULT 0,
    full_themes     IN BOOLEAN DEFAULT FALSE,
    num_themes       IN NUMBER DEFAULT 50);
```

**index_name**
Specify the name of the index for the text column.

**textkey**
Specify the unique identifier (usually the primary key) for the document.

The `textkey` parameter can be one of the following:

- a single column primary key value
- an encoded specification for a composite (multiple column) primary key. When `textkey` is a composite key, you must encode the composite textkey string using the `CTX_DOC.PKENCODE` procedure.
- the rowid of the row containing the document

You toggle between primary key and rowid identification using `CTX_DOC.SET_KEY_TYPE`. 
restab
You can specify that this procedure store results to either a table or to an in-memory PL/SQL table.

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

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

To store results in an in-memory table, specify the name of the in-memory table of type THEME_TAB. The THEME_TAB datatype is defined as follows:

```sql
type theme_rec is record (    theme varchar2(2000),    weight number );

type theme_tab is table of theme_rec index by binary_integer;
```

CTX_DOC.THEMES clears the THEME_TAB you specify before the operation.

query_id
Specify the identifier used to identify the row(s) inserted into restab.

full_themes
Specify whether this procedure generates a single theme or a hierarchical list of parent themes (full themes) for each document theme.

Specify TRUE for this procedure to write full themes to the THEME column of the result table.

Specify FALSE for this procedure to write single theme information to the THEME column of the result table. This is the default.

num_themes
Specify the maximum number of themes to retrieve. For example, if you specify 10, up to first 10 themes are returned for the document. The default is 50.

If you specify 0 or NULL, this procedure returns all themes in a document. If the document contains more than 50 themes, only the first 50 themes show conceptual hierarchy.
**Examples**

**In-Memory Themes**
The following example generates the first 10 themes for document 1 and stores them in an in-memory table called `the_themes`. The example then loops through the table to display the document themes.

```sql
declare
    the_themes ctx_doc.theme_tab;
begin
    ctx_doc.themes('myindex','1',the_themes, numthemes=>10);
    for i in 1..the_themes.count loop
        dbms_output.put_line(the_themes(i).theme||':'||the_themes(i).weight);
    end loop;
end;
```

**Theme Table**
The following example creates a theme table called `CTX_THEMES`:

```sql
create table CTX_THEMES (query_id number,
    theme varchar2(2000),
    weight number);
```

**Single Themes**
To obtain a list of up to the first 20 themes where each element in the list is a single theme, issue a statement like the following:

```sql
begin
    ctx_doc.themes('newsindex','34','CTX_THEMES',1, full_themes => FALSE,
        num_themes=> 20);
end;
```

**Full Themes**
To obtain a list of the top 20 themes where each element in the list is a hierarchical list of parent themes, issue a statement like the following:

```sql
begin
    ctx_doc.themes('newsindex','34','CTX_THEMES',1, full_themes => TRUE, num_themes=>20); 
end;
```
Use this procedure to identify all text tokens in a document. The tokens returned are those tokens which are inserted into the index. This feature is useful for implementing document classification, routing, or clustering. Stopwords are not returned. Section tags are not returned because they are not text tokens.

Syntax 1: In-Memory Table Storage

```sql
CTX_DOC.TOKENS(index_name IN VARCHAR2,
textkey IN VARCHAR2,
restab IN OUT NOCOPY TOKEN_TAB);
```

Syntax 2: Result Table Storage

```sql
CTX_DOC.TOKENS(index_name IN VARCHAR2,
textkey IN VARCHAR2,
restab IN VARCHAR2,
query_id IN NUMBER DEFAULT 0);
```

**index_name**
Specify the name of the index for the text column.

**textkey**
Specify the unique identifier (usually the primary key) for the document.

The `textkey` parameter can be one of the following:

- a single column primary key value
- encoded specification for a composite (multiple column) primary key. To encode a composite `textkey`, use the `CTX_DOC.PKENCODE` procedure.
- the rowid of the row containing the document

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

**restab**
You can specify that this procedure store results to either a table or to an in-memory PL/SQL table.
The tokens returned are those tokens which are inserted into the index for the document (or row) named with textkey. Stop words are not returned. Section tags are not returned because they are not text tokens.

**Specifying a Token Table**
To store results to a table, specify the name of the table. Token tables can be named anything, but must include the following columns, with names and data types as specified.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_ID</td>
<td>NUMBER</td>
<td>The identifier for the results generated by a particular call to CTX_DOC.TOKENS (only populated when table is used to store results from multiple TOKEN calls)</td>
</tr>
<tr>
<td>TOKEN</td>
<td>VARCHAR2(64)</td>
<td>The token string in the text.</td>
</tr>
<tr>
<td>OFFSET</td>
<td>NUMBER</td>
<td>The position of the token in the document, relative to the start of document which has a position of 1.</td>
</tr>
<tr>
<td>LENGTH</td>
<td>NUMBER</td>
<td>The character length of the token.</td>
</tr>
</tbody>
</table>

**Specifying an In-Memory Table**
To store results to an in-memory table, specify the name of the in-memory table of type TOKEN_TAB. The TOKEN_TAB datatype is defined as follows:

```plsql
type token_rec is record (
    token varchar2(64),
    offset number,
    length number
);

type token_tab is table of token_rec index by binary_integer;

CTX_DOC.TOKENS clears the TOKEN_TAB you specify before the operation.

query_id
Specify the identifier used to identify the row(s) inserted into restab.
Examples

**In-Memory Tokens**
The following example generates the tokens for document 1 and stores them in an in-memory table, declared as `the_tokens`. The example then loops through the table to display the document tokens.

declare
    the_tokens ctx_doc.token_tab;

begin
    ctx_doc.tokens('myindex','1',the_tokens);
    for i in 1..the_tokens.count loop
        dbms_output.put_line(the_tokens(i).token);
    end loop;
end;
This chapter provides reference information for using the CTX_OUTPUT PL/SQL package.

CTX_OUTPUT contains the following stored procedures:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_EVENT</td>
<td>Add an event to the index log.</td>
</tr>
<tr>
<td>ADD_TRACE</td>
<td>Enable tracing.</td>
</tr>
<tr>
<td>END_LOG</td>
<td>Halt logging of index and document services requests.</td>
</tr>
<tr>
<td>END_QUERY_LOG</td>
<td>Stop logging queries into a logfile.</td>
</tr>
<tr>
<td>GET_TRACE_VALUE</td>
<td>Return the value of a trace.</td>
</tr>
<tr>
<td>LOG_TRACES</td>
<td>Print traces to logfile.</td>
</tr>
<tr>
<td>LOGFILENAME</td>
<td>Return the name of the current log file.</td>
</tr>
<tr>
<td>REMOVE_EVENT</td>
<td>Remove an event from the index log.</td>
</tr>
<tr>
<td>REMOVE_TRACE</td>
<td>Disable tracing.</td>
</tr>
<tr>
<td>RESET_TRACE</td>
<td>Clear a trace.</td>
</tr>
<tr>
<td>START_LOG</td>
<td>Start logging index and document service requests.</td>
</tr>
<tr>
<td>START_QUERY_LOG</td>
<td>Create a log file of queries.</td>
</tr>
</tbody>
</table>
ADD_EVENT

Use this procedure to add an event to the index log for more detailed log output.

Syntax

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

**event**

Specify the type of index event to log. You can add the following events:

- `CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID`, which logs the rowid of each row after it is indexed. This is useful for debugging a failed index operation.
- `CTX_OUTPUT.EVENT_OPT_PRINT_TOKEN`, which prints each token as it is being optimized.

Example

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

Use this procedure to enable a trace. If the trace has not been enabled, this call adds
the trace to the list of active traces and resets its value to 0. If the trace has already
been enabled, an error is raised.

Syntax

```
CTX_OUTPUT.ADD_TRACE(trace_id BINARY_INTEGER);
```

`trace_id`
Specify the ID of the trace to enable. See Table 9–1 for possible trace values.

Notes

Table 9–1 shows the available traces:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>ID</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDX_USER_DATASTORE</td>
<td>1</td>
<td>time spent executing user datastore</td>
</tr>
<tr>
<td>IDX_INSO_FILTER</td>
<td>2</td>
<td>time spent invoking the INSO filter</td>
</tr>
<tr>
<td>QRY_XX_TIME</td>
<td>3</td>
<td>time spent executing the $X cursor</td>
</tr>
<tr>
<td>QRY_XF_TIME</td>
<td>4</td>
<td>time spent fetching from $X</td>
</tr>
<tr>
<td>QRY_X_ROWS</td>
<td>5</td>
<td>total number of rows whose token metadata was fetched from $X</td>
</tr>
<tr>
<td>QRY_IF_TIME</td>
<td>6</td>
<td>time spent fetching the LOB locator from $I</td>
</tr>
<tr>
<td>QRY_IR_TIME</td>
<td>7</td>
<td>time spent reading $I LOB information</td>
</tr>
<tr>
<td>QRY_I_ROWS</td>
<td>8</td>
<td>number of rows whose $I token_info was actually read</td>
</tr>
<tr>
<td>QRY_I_SIZE</td>
<td>9</td>
<td>number of bytes read from $I LOBs</td>
</tr>
<tr>
<td>QRY_R_TIME</td>
<td>10</td>
<td>time spent fetching and reading $R information</td>
</tr>
<tr>
<td>QRY_CON_TIME</td>
<td>11</td>
<td>time spent in CONTAINS processing (drexrcontains/drexrstart/drexrfetch)</td>
</tr>
</tbody>
</table>

Tracing is independent of logging. Logging does not have to be on to start tracing,
and vice-versa.
Related Topics

See Also: "REMOVE_TRACE" on page 9-11, "LOG_TRACES" on page 9-8, and "RESET_TRACE" on page 9-12, as well as the Oracle Text Application Developer’s Guide
END_LOG

Halt logging index and document service requests

Syntax

```
CTX_OUTPUT.END_LOG;
```

Example

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

Use this procedure to stop logging queries into a logfile created with CTX_OUTPUT.START_QUERY_LOG.

Syntax

CTX_OUTPUT.END_QUERY_LOG;

Example

begin
  CTX_OUTPUT.START_QUERY_LOG('mylog1');
  < get queries >
  CTX_OUTPUT.END_QUERY_LOG;
end;
GET_TRACE_VALUE

Use this procedure to programmatically retrieve the current value of a trace.

Syntax

```sql
CTX_OUTPUT.GET_TRACE_VALUE(trace_id BINARY_INTEGER);
```

**trace_id**
Specify the trace ID whose value you want. See Table 9–1, "Available Traces" on page 9-3 for possible values.

Example

This sets the value of the variable `value`:

```sql
value := ctx_output.get_trace_value(trace_id);
```

Notes

You can also retrieve trace values through SQL:

```sql
select * from ctx_trace_values;
```

See "CTX_TRACE_VALUES" on page G-13 for the entries in the `CTX_TRACE_VALUES` view.

If the trace has not been enabled, an error is raised.

Traces are not reset to 0 by this call.

Related Topics

See Also: ADD_TRACE on page 9-3 and the Oracle Text Application Developer’s Guide
LOG_TRACES

Use this procedure to print all active traces to the logfile.

Syntax

CTX_OUTPUT.LOG_TRACES;

Notes

If logging has not been started, an error is raised.
Traces are not reset to 0 by this call.

This procedure looks for the logfile in the directory specified by the LOG_DIRECTORY system parameter, which is $ORACLE_HOME/ctx/log on UNIX. You can query the CTX_PARAMETERS view to find the current setting.

Related Topics

See Also: ADD_TRACE on page 9-3 and the Oracle Text Application Developer’s Guide
LOGFILENAME

Returns the filename for the current log. This procedure looks for the logfile in the directory specified by the LOG_DIRECTORY system parameter, which is $ORACLE_HOME/ctx/log on UNIX. You can query the CTX_PARAMETERS view to find the current setting.

Syntax

```
CTX_OUTPUT.LOGFILENAME RETURN VARCHAR2;
```

Returns

Log file name.

Example

```
declare
    logname varchar2(100);
begin
    logname := CTX_OUTPUT.LOGFILENAME;
    dbms_output.put_line('The current log file is: ' || logname);
end;
```
Use this procedure to remove an event from the index log.

**Syntax**

```sql
CTX_OUTPUT.REMOVE_EVENT(event in varchar2);
```

**event**

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

**Example**

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

Use this procedure to disable a trace.

Syntax

```sql
CTX_OUTPUT.REMOVE_TRACE(trace_id BINARY_INTEGER);
```

**trace_id**
Specify the ID of the trace to disable. See Table 9–1, "Available Traces" on page 9-3 for possible values.

Notes
If the trace has not been enabled, an error is raised.

Related Topics

See Also: ADD_TRACE on page 9-3 and the Oracle Text Application Developer’s Guide
Use this procedure to clear a trace (that is, reset it to 0).

**Syntax**

```sql
CTX_OUTPUT.RESET_TRACE(trace_id BINARY_INTEGER);
```

**trace_id**

Specify the ID of the trace to reset. See Table 9–1, “Available Traces” on page 9-3 for possible values.

**Notes**

If the trace has not been enabled, an error is raised.

**Related Topics**

See Also: ADD_TRACE on page 9-3 and the Oracle Text Application Developer’s Guide
START_LOG

Begin logging index and document service requests.

Syntax

```sql
CTX_OUTPUT.START_LOG(logfile in varchar2, overwrite in default true);
```

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

**overwrite**
Specify whether you want to overwrite or append to the original query log file specified by `logfile`, if it already exists. The default is to overwrite the original query log file.

Example

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

Notes

Logging is independent of tracing. Logging does not have to be on to start tracing, and vice-versa.
Begin logging query requests into a query log file.

Use `CTX_OUTPUT.END_QUERY_LOG` to stop logging queries. Use `CTX_REPORT.QUERY_LOG_SUMMARY` to obtain reports on logged queries, such as which queries returned successfully the most times.

The query log includes the query string, the index name, and the timestamp of the query, as well as whether or not the query successfully returned a hit. A successful query for the phrase "Blues Guitarists" made at 6:46 (local time) on November 11th, 2003, would be entered into the query log in this form:

```xml
<QuerySet><TimeStamp>18:46:51 02/04/03</TimeStamp><IndexName>IDX_SEARCH_TABLE</IndexName><Query>Blues Guitarists</Query><ReturnHit>Yes</ReturnHit></QuerySet>
```

**Syntax**

`CTX_OUTPUT.START_QUERY_LOG(logfile in varchar2, overwrite in default true);`

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

- **overwrite**
  Specify whether you want to overwrite or append to the original query log file specified by `logfile`, if it already exists. The default is to overwrite the original query log file.

**Example**

```sql
begin
    CTX_OUTPUT.START_QUERY_LOG('mylog1');
    /* get queries */
    CTX_OUTPUT.END_QUERY_LOG;
end;
```
This chapter describes the `CTX_QUERY` PL/SQL package you can use for generating query feedback, counting hits, and creating stored query expressions.

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

The `CTX_QUERY` package includes the following procedures and functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROWSE_WORDS</td>
<td>Returns the words around a seed word in the index.</td>
</tr>
<tr>
<td>COUNT_HITS</td>
<td>Returns the number hits to a query.</td>
</tr>
<tr>
<td>EXPLAIN</td>
<td>Generates query expression parse and expansion information.</td>
</tr>
<tr>
<td>HFEEDBACK</td>
<td>Generates hierarchical query feedback information (broader term, narrower term, and related term).</td>
</tr>
<tr>
<td>REMOVE_SQE</td>
<td>Removes a specified stored query expression from the SQL tables.</td>
</tr>
<tr>
<td>STORE_SQE</td>
<td>Executes a query and stores the results in stored query expression tables.</td>
</tr>
</tbody>
</table>
This procedure enables you to browse words in an Oracle Text index. You specify a seed word and BROWSE_WORDS returns the words around it in the index, and an approximate 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

```sql
ctx_query.browse_words(
    index_name IN VARCHAR2,
    seed IN VARCHAR2,
    restab IN VARCHAR2,
    browse_id IN NUMBER DEFAULT 0,
    numwords IN NUMBER DEFAULT 10,
    direction IN VARCHAR2 DEFAULT BROWSE_AROUND,
    part_name IN VARCHAR2 DEFAULT NULL
);
```

Syntax 2: To Store Results in Memory

```sql
ctx_query.browse_words(
    index_name IN VARCHAR2,
    seed IN VARCHAR2,
    resarr IN OUT BROWSE_TAB,
    numwords IN NUMBER DEFAULT 10,
    direction IN VARCHAR2 DEFAULT BROWSE_AROUND,
    part_name IN VARCHAR2 DEFAULT NULL
);
```

index
Specify the name of the index. You can specify schema.name. Must be a local index.
seed
Specify the seed word. This word is lexed before browse expansion. The word need not exist in the token table. seed must be a single word. Using multiple words as the seed will result in an error.

restab
Specify the name of the result table. You can enter restab as schema.name. The table must exist before you call this procedure, and you must have INSERT permissions on the table. This table must have the following schema.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>browse_id</td>
<td>number</td>
</tr>
<tr>
<td>word</td>
<td>varchar2(64)</td>
</tr>
<tr>
<td>doc_count</td>
<td>number</td>
</tr>
</tbody>
</table>

Existing rows in restab are not deleted before BROWSE_WORDS is called.

resarr
Specify the name of the result array. resarr is of type ctx_query.browse_tab.

type browse_rec is record (  
  word varchar2(64),  
  doc_count number  
);  
type browse_tab is table of browse_rec index by binary_integer;

browse_id
Specify a numeric identifier between 0 and 2^{32}. The rows produced for this browse have a value of in the browse_id column in restab. When you do not specify browse_id, it defaults to 0.

numwords
Specify the number of words returned.

direction
Specify the direction for the browse. You can specify one of:

<table>
<thead>
<tr>
<th>value</th>
<th>behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEFORE</td>
<td>Browse seed word and words alphabetically before the seed.</td>
</tr>
</tbody>
</table>
BROWSE_WORDS

<table>
<thead>
<tr>
<th>value</th>
<th>behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>AROUND</td>
<td>Browse seed word and words alphabetically before and after the seed.</td>
</tr>
<tr>
<td>AFTER</td>
<td>Browse seed word and words alphabetically after the seed.</td>
</tr>
</tbody>
</table>

Symbols CTX_QUERY.BROWSE_BEFORE, CTX_QUERY.BROWSE_AROUND, and CTX_QUERY.BROWSE_AFTER are defined for these literal values as well.

part_name
Specify the name of the index partition to browse.

Example

Browsing Words with Result Table
begin
ctx_query.browse_words('myindex','dog','myres',numwords=>5,direction=>'AROUND');
end;

select word, doc_count from myres order by word;

<table>
<thead>
<tr>
<th>WORD</th>
<th>DOC_COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CZAR</td>
<td>15</td>
</tr>
<tr>
<td>DARLING</td>
<td>5</td>
</tr>
<tr>
<td>DOC</td>
<td>73</td>
</tr>
<tr>
<td>DUNK</td>
<td>100</td>
</tr>
<tr>
<td>EAR</td>
<td>3</td>
</tr>
</tbody>
</table>

Browsing Words with Result Array
set serveroutput on;
declare
  resarr ctx_query.browse_tab;
begin
ctx_query.browse_words('myindex','dog',resarr,5,CTX_QUERY.BROWSE_AROUND);
for i in 1..resarr.count loop
  dbms_output.put_line(resarr(i).word || ':' || resarr(i).doc_count);
end loop;
end;
COUNT_HITS

Returns the number of hits for the specified query. You can call COUNT_HITS in exact or estimate mode. Exact mode returns the exact number of hits for the query. Estimate mode returns an upper-bound estimate but runs faster than exact mode.

Syntax

```
CTX_QUERY.COUNT_HITS (  
    index_name     IN VARCHAR2,  
    text_query     IN VARCHAR2,  
    exact          IN BOOLEAN  DEFAULT TRUE,  
    part_name      IN VARCHAR2 DEFAULT NULL  
) RETURN NUMBER;
```

**index_name**
Specify the index name.

**text_query**
Specify the query.

**exact**
Specify TRUE for an exact count. Specify FALSE for an upper-bound estimate.

Specifying FALSE returns a less accurate number but runs faster. Specifying FALSE might return a number which is too high if rows have been updated or deleted since the last FULL index optimize. Optimizing in full mode removes these false hits, and then exact set to FALSE will return the same number as exact set to TRUE.

**part_name**
Specify the name of the index partition to query.

**Notes**

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

If the index was created with the TRANSACTIONAL parameter, then COUNT_HITS will include pending rowids as well as those that have been synchronized.
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 Text evaluates a query is useful for refining and debugging queries. You can also design your application so that it uses the explain plan information to help users write better queries.

Limitation

You cannot use EXPLAIN with remote queries.

Syntax

```
CTX_QUERY.EXPLAIN(
    index_name IN VARCHAR2,
    text_query IN VARCHAR2,
    explain_table IN VARCHAR2,
    sharelevel IN NUMBER DEFAULT 0,
    explain_id IN VARCHAR2 DEFAULT NULL,
    part_name IN VARCHAR2 DEFAULT NULL
));
```

**index_name**

Specify the name of the index to be queried.
text_query
Specify the query expression to be used as criteria for selecting rows.

When you include a wildcard, fuzzy, or soundex operator in text_query, this procedure looks at the index tables to determine the expansion.

Wildcard, fuzzy (?), and soundex (!) expression feedback does not account for lazy deletes as in regular queries.

explain_table
Specify the name of the table used to store representation of the parse tree for text_query. You must have at least INSERT and DELETE privileges on the table used to store the results from EXPLAIN.

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

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

When you specify 0, the system automatically truncates the result table before the next call to EXPLAIN.

When you specify 1 for shared use, this procedure does not truncate the result table. Only results with the same explain_id are updated. When no results with the same explain_id exist, new results are added to the EXPLAIN table.

explain_id
Specify a name that identifies the explain results returned by an EXPLAIN procedure when more than one EXPLAIN call uses the same shared EXPLAIN table. This parameter defaults to NULL.

part_name
Specify the name of the index partition to query.

Example

Creating the Explain Table
To create an explain table called test_explain for example, use the following SQL statement:
create table test_explain(
    explain_id varchar2(30),
    id number,
    parent_id number,
    operation varchar2(30),
    options varchar2(30),
    object_name varchar2(64),
    position number,
    cardinality number);

Executing CTX_QUERY.EXPLAIN
To obtain the expansion of a query expression such as `comp% OR ?smith`, use CTX_QUERY.EXPLAIN as follows:

ctx_query.explain(
    index_name => 'newindex',
    text_query => 'comp% OR ?smith',
    explain_table => 'test_explain',
    sharelevel => 0,
    explain_id => 'Test');

Retrieving Data from Explain Table
To read the explain table, you can select the columns as follows:

select explain_id, id, parent_id, operation, options, object_name, position
from test_explain order by id;

The output is ordered by ID to simulate a hierarchical query:

<table>
<thead>
<tr>
<th>EXPLAIN_ID</th>
<th>ID</th>
<th>PARENT_ID</th>
<th>OPERATION</th>
<th>OPTIONS</th>
<th>OBJECT_NAME</th>
<th>POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>1</td>
<td>0 OR</td>
<td>NULL</td>
<td>COMPTROLLER</td>
<td>SMITH</td>
<td>1</td>
</tr>
<tr>
<td>Test</td>
<td>2</td>
<td>1 EQUIVALENCE</td>
<td>NULL</td>
<td>COMPUTER</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>3</td>
<td>2 WORD</td>
<td>NULL</td>
<td>SMITH</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>4</td>
<td>2 WORD</td>
<td>NULL</td>
<td>SMYTHER</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>5</td>
<td>1 EQUIVALENCE</td>
<td>(?)</td>
<td>SMITH</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>6</td>
<td>5 WORD</td>
<td>NULL</td>
<td>SMITH</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>7</td>
<td>5 WORD</td>
<td>NULL</td>
<td>SMYTHER</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Related Topics

Chapter 3, "CONTAINS Query Operators"
Appendix H, "Stopword Transformations"
In English or French, this procedure generates hierarchical query feedback information (broader term, narrower term, and related term) for the specified query.

Broader term, narrower term, and related term information is obtained from the knowledge base. However, only knowledge base terms that are also in the index are returned as query feedback information. This increases the chances that terms returned from HFEEDBACK produce hits over the currently indexed document set.

Hierarchical query feedback information is useful for suggesting other query terms to the user.

**Note:** CTX_QUERY.HFEEDBACK is only supported in English and French.

**Syntax**

```sql
CTX_QUERY.HFEEDBACK(
    index_name    IN VARCHAR2,
    text_query    IN VARCHAR2,
    feedback_table IN VARCHAR2,
    sharelevel    IN NUMBER DEFAULT 0,
    feedback_id   IN VARCHAR2 DEFAULT NULL,
    part_name     IN VARCHAR2 DEFAULT NULL
);
```

**index_name**
Specify the name of the index for the text column to be queried.

**text_query**
Specify the query expression to be used as criteria for selecting rows.

**feedback_table**
Specify the name of the table used to store the feedback terms.

**See Also:** For more information about the structure of the explain table, see "HFEEDBACK Table" in Appendix A, "Result Tables".
sharelevel
Specify whether feedback_table is shared by multiple HFEEDBACK calls. Specify 0 for exclusive use and 1 for shared use. This parameter defaults to 0 (single-use).

When you specify 0, the system automatically truncates the feedback table before the next call to HFEEDBACK.

When you specify 1 for shared use, this procedure does not truncate the feedback table. Only results with the same feedback_id are updated. When no results with the same feedback_id exist, new results are added to the feedback table.

feedback_id
Specify a value that identifies the feedback results returned by a call to HFEEDBACK when more than one HFEEDBACK call uses the same shared feedback table. This parameter defaults to NULL.

part_name
Specify the name of the index partition to query.

Example

Create HFEEDBACK Result Table
Create a result table to use with CTX_QUERY.HFEEDBACK as follows:

```sql
CREATE TABLE restab ( 
    feedback_id VARCHAR2(30), 
    id NUMBER, 
    parent_id NUMBER, 
    operation VARCHAR2(30), 
    options VARCHAR2(30), 
    object_name VARCHAR2(80), 
    position NUMBER, 
    bt_feedback ctxsys.ctx_feedback_type, 
    rt_feedback ctxsys.ctx_feedback_type, 
    nt_feedback ctxsys.ctx_feedback_type 
) NESTED TABLE bt_feedback STORE AS res_bt 
NESTED TABLE rt_feedback STORE AS res_rt 
NESTED TABLE nt_feedback STORE AS res_nt;
```

CTX_FEEDBACK_TYPE is a system-defined type in the CTXSYS schema.
See Also: For more information about the structure of the HFEEDBACK table, see "HFEEDBACK Table" in Appendix A, "Result Tables".

Call CTX_QUERY.HFEEDBACK
The following code calls the HFEEDBACK procedure with the query computer industry.

```
BEGIN
ctx_query.hfeedback (index_name     => 'my_index',
                     text_query     => 'computer industry',
                     feedback_table => 'restab',
                     sharelevel     => 0,
                     feedback_id    => 'query10'
                   );
END;
```

Select From the Result Table
The following code extracts the feedback data from the result table. It extracts broader term, narrower term, and related term feedback separately from the nested tables.

```
DECLARE
  i NUMBER;
BEGIN
  FOR frec IN (
    SELECT object_name, bt_feedback, rt_feedback, nt_feedback
    FROM restab
    WHERE feedback_id = 'query10' AND object_name IS NOT NULL
  ) LOOP
    dbms_output.put_line('Broader term feedback for ' || frec.object_name || ' : ');
    i := frec.bt_feedback.FIRST;
    WHILE i IS NOT NULL LOOP
      dbms_output.put_line(frec.bt_feedback(i).text);
      i := frec.bt_feedback.NEXT(i);
    END LOOP;
    dbms_output.put_line('Related term feedback for ' || frec.object_name || ' : ');
    i := frec.rt_feedback.FIRST;
    WHILE i IS NOT NULL LOOP
      dbms_output.put_line(frec.rt_feedback(i).text);
  END LOOP;
END;
```

See Also: For more information about the structure of the HFEEDBACK table, see "HFEEDBACK Table" in Appendix A, "Result Tables".
i := frec.rt_feedback.NEXT(i);
END LOOP;

dbms_output.put_line('Narrower term feedback for ' || frec.object_name || ':
');

i := frec.nt_feedback.FIRST;
WHILE i IS NOT NULL LOOP
   dbms_output.put_line(frec.nt_feedback(i).text);
i := frec.nt_feedback.NEXT(i);
END LOOP;

END LOOP;
END;

Sample Output

The following output is for the preceding example, which queries on computer industry:

Broader term feedback for computer industry:
hard sciences
Related term feedback for computer industry:
computer networking
electronics
knowledge
library science
mathematics
optical technology
robotics
satellite technology
semiconductors and superconductors
symbolic logic
telecommunications industry
Narrower term feedback for computer industry:
ABEND - abnormal end of task
AT&T Starlans
ATI Technologies, Incorporated
ActivCard
Actrade International Ltd.
Alta Technology
Amiga Format
Amiga Library Services
Amiga Shopper
Amstrat Action
Note: The HFEEDBACK information you obtain depends on the contents of your index and knowledge base and as such might differ from the sample shown.
The `CTX_QUERY.REMOVE_SQE` procedure removes the specified stored query expression.

**Syntax**

```sql
CTX_QUERY.REMOVE_SQE(query_name IN VARCHAR2);
```

**query_name**
Specify the name of the stored query expression to be removed.

**Examples**

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

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

Supported Operators

Stored query expressions support all of the CONTAINS query operators. Stored query expressions also support all of the special characters and other components that can be used in a query expression, including other stored query expressions.

Privileges

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

Syntax

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

**query_name**
Specify the name of the stored query expression to be created.

**text_query**
Specify the query expression to be associated with query_name.

Examples

```
begin
ctx_query.store_sqe('disasters', 'hurricanes | earthquakes');
end;
```
This chapter describes how to use the CTX_REPORT package to create reports on indexing and querying. These reports can help you troubleshoot problems or fine-tune your applications.

This chapter contains the following topics:

- Procedures in CTX_REPORT
- Using the Function Versions

For an overview of the CTX_REPORT package and how you can use the various procedures described here, see the Oracle Text Application Developer’s Guide.

Procedures in CTX_REPORT

The CTX_REPORT package contains the following procedures:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE_INDEX</td>
<td>Creates a report describing the index.</td>
</tr>
<tr>
<td>DESCRIBE_POLICY</td>
<td>Creates a report describing a policy.</td>
</tr>
<tr>
<td>CREATE_INDEX_SCRIPT</td>
<td>Creates a SQL*Plus script to duplicate the named index.</td>
</tr>
<tr>
<td>CREATE_POLICY_SCRIPT</td>
<td>Creates a SQL*Plus script to duplicate the named policy.</td>
</tr>
<tr>
<td>INDEX_SIZE</td>
<td>Creates a report to show the internal objects of an index, their tablespaces and used sizes.</td>
</tr>
<tr>
<td>INDEX_STATS</td>
<td>Creates a report to show the various statistics of an index.</td>
</tr>
<tr>
<td>QUERY_LOG_SUMMARY</td>
<td>Creates a report showing query statistics</td>
</tr>
</tbody>
</table>
Using the Function Versions

Some of the procedures in the CTX_REPORT package have function versions. You can call these functions as follows:

```sql
select ctx_report.describe_index('MYINDEX') from dual;
```

In SQL*Plus, to generate an output file to send to support, you can do:

```sql
set long 64000
set pages 0
set heading off
set feedback off
spool outputfile
select ctx_report.describe_index('MYINDEX') from dual;
spool off
```

### Name Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN_INFO</td>
<td>Creates a report showing the information for a token, decoded.</td>
</tr>
<tr>
<td>TOKEN_TYPE</td>
<td>Translates a name and returns a numeric token type.</td>
</tr>
</tbody>
</table>
DESCRIBE_INDEX

Creates a report describing the index. This includes the settings of the index metadata, the indexing objects used, the settings of the attributes of the objects, and index partition descriptions, if any.

You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

Syntax

```plsql
procedure CTX_REPORT.DESCRIBE_INDEX(
    index_name     IN VARCHAR2,
    report         IN OUT NOCOPY CLOB,
    report_format  IN VARCHAR2 DEFAULT FMT_TEXT
);

function CTX_REPORT.DESCRIBE_INDEX(
    index_name     IN VARCHAR2,
    report_format  IN VARCHAR2 DEFAULT FMT_TEXT
) return CLOB;
```

**index_name**
Specify the name of the index to describe.

**report**
Specify the CLOB locator to which to write the report.

If `report` is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The `report` CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

**report_format**
Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX_REPORT.FMT_TEXT or CTX_REPORT.FMT_XML.
DESCRIBE_POLICY

Creates a report describing the policy. This includes the settings of the policy metadata, the indexing objects used, the settings of the attributes of the objects.

You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

Syntax

procedure CTX_REPORT.DESCRIBE_POLICY(
    policy_name    IN VARCHAR2,
    report         IN OUT NOCOPY CLOB,
    report_format  IN VARCHAR2 DEFAULT FMT_TEXT
);

function CTX_REPORT.DESCRIBE_POLICY(
    policy_name    IN VARCHAR2,
    report_format  IN VARCHAR2 DEFAULT FMT_TEXT
) return CLOB;

report
Specify the CLOB locator to which to write the report.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

report_format
Specify whether the report should be generated as ‘TEXT’ or as ‘XML’. TEXT is the default. You can also specify the values CTX_REPORT.FMT_TEXT or CTX_REPORT.FMT_XML.

policy_name
Specify the name of the policy to describe
CREATE_INDEX_SCRIPT

Creates a SQL*Plus script which will create a text index that duplicates the named text index.

The created script will include creation of preferences identical to those used in the named text index. However, the names of the preferences will be different.

You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

Syntax

procedure CTX_REPORT.CREATE_INDEX_SCRIPT(
   index_name      in varchar2,
   report          in out nocopy clob,
   prefname_prefix in varchar2 default null
);

function CTX_REPORT.CREATE_INDEX_SCRIPT(
   index_name      in varchar2,
   prefname_prefix in varchar2 default null
) return clob;

index_name
Specify the name of the index.

report
Specify the CLOB locator to which to write the script.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller’s responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

prefname_prefix
Specify optional prefix to use for preference names.

If prefname_prefix is omitted or NULL, index name will be used. The prefname_prefix follows index length restrictions.
CREATE_POLICY_SCRIPT

 Creates a SQL*Plus script which will create a text policy that duplicates the named text policy.

 The created script will include creation of preferences identical to those used in the named text policy.

 You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

 Syntax

 procedure CTX_REPORT.CREATE_POLICY_SCRIPT(
     policy_name      in varchar2,
     report           in out nocopy clob,
     prefname_prefix  in varchar2 default null
 )

 function CTX_REPORT.CREATE_POLICY_SCRIPT(
     policy_name      in varchar2,
     prefname_prefix  in varchar2 default null
 ) return clob;

 policy_name
 Specify the name of the policy.

 report
 Specify the locator to which to write the script.

 If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller’s responsibility to free this temporary CLOB as needed.

 The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

 prefname_prefix
 Specify the optional prefix to use for preference names. If prefname_prefix is omitted or NULL, policy name will be used. prefname_prefix follows policy length restrictions.
INDEX_SIZE

Creates a report showing the internal objects of the text index or text index partition, and their tablespaces, allocated, and used sizes.

You can call this operation as a procedure with an IN OUT CLOB parameter, or as a function that returns the report as a CLOB.

Syntax

procedure CTX_REPORT.INDEX_SIZE(
    index_name     IN VARCHAR2,
    report         IN OUT NOCOPY CLOB,
    part_name      IN VARCHAR2 DEFAULT NULL,
    report_format  IN VARCHAR2 DEFAULT FMT_TEXT
);

function CTX_REPORT.INDEX_SIZE(
    index_name     IN VARCHAR2,
    part_name      IN VARCHAR2 DEFAULT NULL,
    report_format  IN VARCHAR2 DEFAULT FMT_TEXT
) return clob;

index_name
Specify the name of the index to describe

report
Specify the CLOB locator to which to write the report.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call

part_name
Specify the name of the index partition (optional). If part_name is NULL, and the index is a local partitioned text index, then all objects of all partitions will be displayed. If part_name is provided, then only the objects of a particular partition will be displayed.
**report_format**
Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX_REPORT.FMT_TEXT or CTX_REPORT.FMT_XML.
INDEX_STATS

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

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

```sql
procedure index_stats(
    index_name     IN VARCHAR2,
    report         IN OUT NOCOPY CLOB,
    part_name      IN VARCHAR2 DEFAULT NULL,
    frag_stats     IN BOOLEAN DEFAULT TRUE,
    list_size      IN NUMBER DEFAULT 100,
    report_format  IN VARCHAR2 DEFAULT FMT_TEXT
);
```

**index_name**
Specify the name of the index to describe. This must be a CONTEXT index.

**report**
Specify the CLOB locator to which to write the report. If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller’s responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

**part_name**
Specify the name of the index partition. If the index is a local partitioned index, then part_name must be provided. INDEX_STATS will calculate the statistics for that index partition.

**frag_stats**
Specify TRUE to calculate fragmentation statistics. If frag_stats is FALSE, the report will not show any statistics relating to size of index data. However, the operation should take less time and resources to calculate the token statistics.

**list_size**
Specify the number of elements in each compiled list. list_size has a maximum value of 1000.
**report_format**
Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX_REPORT.FMT_TEXT or CTX_REPORT.FMT_XML.

**Example**

Here's an example of using CTX_REPORT.INDEX_STATS:

```sql
create table output (result CLOB);

declare
    x clob := null;
begin
    ctx_report.index_stats('tdrbprx21',x);
    insert into output values (x);
    commit;
    dbms_lob.freetemporary(x);
end;
/
```

```sql
set long 32000
set head off
set pagesize 10000
select * from output;
```

The following is sample output for INDEX_STATS on a context index. This report has been truncated for clarity. It shows some of the token statistics and all of the fragmentation statistics.

The fragmentation statistics are at the end of the report. It tells you optimal row fragmentation, an estimated amount of garbage data in the index, and a list of the most fragmented tokens. Running CTX_DDL.OPTIMIZE_INDEX cleans up the index.

```
STATISTICS FOR "DR_TEST"."TDRBPRX21"

indexed documents: 53
allocated docids: 68
$I rows: 16,259
```
TOKEN STATISTICS

unique tokens: 13,445
average $I$ rows for each token: 1.21

tokens with most $I$ rows:
- telecommunications industry (THEME) 6
- science and technology (THEME) 6
- EMAIL (FIELD SECTION "SOURCE") 6
- DEC (FIELD SECTION "TIMESTAMP") 6
- electronic mail (THEME) 6
- computer networking (THEME) 6
- communications (THEME) 6
- 95 (FIELD SECTION "TIMESTAMP") 6
- 15 (FIELD SECTION "TIMESTAMP") 6
- HEADLINE (ZONE SECTION) 6

average size for each token: 8

tokens with largest size:
- T (NORMAL) 405
- SAID (NORMAL) 313
- HEADLINE (ZONE SECTION) 272
- NEW (NORMAL) 267
- I (NORMAL) 230
- MILLION (PREFIX) 222
- D (NORMAL) 219
- MILLION (NORMAL) 215
- U (NORMAL) 192
- DEC (FIELD SECTION "TIMESTAMP") 186

average frequency for each token: 2.00

most frequent tokens:
- HEADLINE (ZONE SECTION) 68
- DEC (FIELD SECTION "TIMESTAMP") 62
- 95 (FIELD SECTION "TIMESTAMP") 62
- 15 (FIELD SECTION "TIMESTAMP") 62
- T (NORMAL) 61
- D (NORMAL) 59
- 881115 (THEME) 58
- 881115 (NORMAL) 58
- I (NORMAL) 55
- geography (THEME) 52

token statistics by type:
- token type: NORMAL
The fragmentation portion of this report is as follows:

<table>
<thead>
<tr>
<th>Fragmentation Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>total size of $I$ data:</td>
</tr>
</tbody>
</table>
$I$ rows:  16,259
estimated $I$ rows if optimal: 13,445
estimated row fragmentation: 17%
garbage docids: 15
estimated garbage size: 21,379 (20.88 KB)

most fragmented tokens:
  telecommunications industry (THEME) 83%
  science and technology (THEME) 83%
  EMAIL (FIELD SECTION "SOURCE") 83%
  DEC (FIELD SECTION "TIMESTAMP") 83%
  electronic mail (THEME) 83%
  computer networking (THEME) 83%
  communications (THEME) 83%
  95 (FIELD SECTION "TIMESTAMP") 83%
  HEADLINE (ZONE SECTION) 83%
  15 (FIELD SECTION "TIMESTAMP") 83%
Obtain a report of logged queries. 

_QUERY_LOG_SUMMARY enables you to analyze queries you have logged. For example, suppose you have an application that searches a database of large animals, and your analysis of queries against it shows that users are continually searching for the word *mouse*; this analysis might induce you to rewrite your application so that a search for *mouse* redirects the user to a database for small animals instead of simply returning an unsuccessful search.

With query analysis, you can find out

- which queries were made
- which queries were successful
- which queries were unsuccessful
- how many times each query was made

You can combine these factors in various ways, such as determining the 50 most frequent unsuccessful queries made by your application.

Query logging is begun with CTX_OUTPUT.START_QUERY_LOG and terminated with CTX_OUTPUT.END_QUERY_LOG.

---

**Note:** You must connect as CTXSYS to use CTX_REPORT.QUERY_LOG_SUMMARY.

---

**See Also:** START_QUERY_LOG and END_QUERY_LOG in Chapter 9, "CTX_OUTPUT Package".

---

**Syntax**

```sql
procedure CTX_REPORT.QUERY_LOG_SUMMARY(  logfile  IN VARCHAR2,
  indexname  IN VARCHAR2 DEFAULT NULL,
  result_table  IN OUT NOCOPY QUERY_TABLE,
  row_num  IN NUMBER,
  most_freq  IN BOOLEAN DEFAULT TRUE,
  has_hit  IN BOOLEAN DEFAULT TRUE
  )
```
logfile
Specify the name of the logfile that contains the queries.

indexname
Specify the name of the context index for which you want the summary report. If you specify NULL, the procedure provides a summary report for all context indexes.

result_table
Specify the name of the in-memory table of type TABLE OF RECORD where the results of the QUERY_LOG_SUMMARY are to go. The default is the location specified by the system parameter LOG_DIRECTORY.

row_num
The number of rows of results from QUERY_LOG_SUMMARY to be reported into the table named by restab. For example, if this is number is 10, most_freq is TRUE, and has_hit is TRUE, then the procedure returns the 10 most frequent queries that were successful (that is, returned hits).

most_freq
Specify whether QUERY_LOG_SUMMARY should return the most frequent or least frequent queries. The default is most frequent queries. If most_freq is set to FALSE, the procedure returns the least successful queries.

has_hit
Specify whether QUERY_LOG_SUMMARY should return queries that are successful (that is, that generate hits) or unsuccessful queries. The default is to count successful queries; set has_hit to FALSE to return unsuccessful queries.

Example
The following example shows how a query log can be used.

First connect as CTXSYS. Then create and populate two tables, and then create an index for each:

```sql
create table qlogtabl (tk number primary key, text varchar2(2000));
insert into qlogtabl values(1, 'The Roman name for France was Gaul.');
insert into qlogtabl values(2, 'The Tour de France is held each summer.');
insert into qlogtabl values(3, 'Jacques Anatole Thibault took the pen name Anatole France.');
create index idx_qlog1 on qlogtabl(text) indextype is ctxsys.context;
create table qlogtab2 (tk number primary key, text varchar2(2000));
insert into qlogtab2 values(1, 'The Great Wall of China is about 2400 kilometers long');
```
insert into qlogtab2 values(2, 'Soccer dates back at least to 217 C.E.');
insert into qlogtab2 values(3, 'The Corn Palace is a tourist attraction in South Dakota.');
create index idx_qlog2 on qlogtab2(text) indextype is ctxsys.context;

**Turn on query logging, creating a log called** `query_log`

exec ctx_output.start_query_log('query.log');

**Now make some queries (some of which will be unsuccessful):**

```sql
select text from qlogtab1 where contains(text, 'France',1)>0;
select text from qlogtab1 where contains(text, 'cheese',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizard',1)>0;
select text from qlogtab2 where contains(text, 'Corn Palace',1)>0;
select text from qlogtab2 where contains(text, 'China',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizards',1)>0;
select text from qlogtab2 where contains(text, 'South Dakota',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizard',1)>0;
select text from qlogtab2 where contains(text, 'China',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizard',1)>0;
select text from qlogtab2 where contains(text, 'company',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizard',1)>0;
select text from qlogtab1 where contains(text, 'France',1)>0;
select text from qlogtab2 where contains(text, 'database',1)>0;
select text from qlogtab1 where contains(text, 'high-tech',1)>0;
select text from qlogtab1 where contains(text, 'database',1)>0;
select text from qlogtab1 where contains(text, 'France',1)>0;
select text from qlogtab1 where contains(text, 'Japan',1)>0;
select text from qlogtab1 where contains(text, 'Egypt',1)>0;
select text from qlogtab1 where contains(text, 'Argentina',1)>0;
select text from qlogtab1 where contains(text, 'Argentina',1)>0;
select text from qlogtab1 where contains(text, 'Argentina',1)>0;
select text from qlogtab1 where contains(text, 'Japan',1)>0;
select text from qlogtab1 where contains(text, 'Egypt',1)>0;
select text from qlogtab1 where contains(text, 'Air Shuttle',1)>0;
select text from qlogtab1 where contains(text, 'Argentina',1)>0;
```

**With the querying over, turn query logging off:**

exec ctx_output.end_query_log;
Use `QUERY_LOG_SUMMARY` to get query reports. In the first instance, you ask to see the three most frequent queries that return successfully. First declare the results table (`the_queries`).

```
set serveroutput on;
declare
    the_queries ctx_report.query_table;
begin
    ctx_report.query_log_summary('query.log', null, the_queries,
        row_num=>3, most_freq=>TRUE, has_hit=>TRUE);
dbms_output.put_line('The 3 most frequent queries returning hits');
dbms_output.put_line('number of times  query string');
for i in 1..the_queries.count loop
    dbms_output.put_line(the_queries(i).times||'                '|the_queries(i).query);
end loop;
end;
/
```

This returns the following:

```
The 3 most frequent queries returning hits
number of times  query string
3                France
2                China
1                Corn Palace
```

Next, look for the three most frequent queries on `idx_qlog1` that were successful.

```
declare
    the_queries ctx_report.query_table;
begin
    ctx_report.query_log_summary('query.log', 'idx_qlog1', the_queries,
        row_num=>3, most_freq=>TRUE, has_hit=>TRUE);
dbms_output.put_line('The 3 most frequent queries returning hits for index idx_qlog1');
dbms_output.put_line('number of times  query string');
for i in 1..the_queries.count loop
    dbms_output.put_line(the_queries(i).times||'                '|the_queries(i).query);
end loop;
end;
/
```
Because only the queries for France were successful, `ctx_report.query_log_summary` returns the following:

The 3 most frequent queries returning hits for index idx_qlog1

<table>
<thead>
<tr>
<th>number of times</th>
<th>query string</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>France</td>
</tr>
</tbody>
</table>

Lastly, ask to see the three least frequent queries that returned no hits (that is, queries that were unsuccessful and called infrequently). In this case, you are interested in queries on both context indexes, so you set the indexname parameter to NULL.

```sql
declare
    the_queries ctx_report.query_table;
begin
    ctx_report.query_log_summary('query.log', null, the_queries, row_num=>3,
        most_freq=>FALSE, has_hit=>FALSE);
    dbms_output.put_line('The 3 least frequent queries returning no hit');
    dbms_output.put_line('number of times  query string');
    for i in 1..the_queries.count loop
        dbms_output.put_line(the_queries(i).times||'                '||the_queries(i).query);
    end loop;
end;
/
```

This returns the following:

The 3 least frequent queries returning no hit

<table>
<thead>
<tr>
<th>number of times</th>
<th>query string</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>high-tech</td>
</tr>
<tr>
<td>1</td>
<td>company</td>
</tr>
<tr>
<td>1</td>
<td>cheese</td>
</tr>
</tbody>
</table>

Argentina and Japan do not make this list, because they are queried more than once, while Corn Palace does not make this list because it is successfully queried.
TOKEN_INFO

Creates a report showing the information for a token, decoded. This procedure will fully scan the info for a token, so it may take a long time to run for really large tokens.

You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

Syntax

```sql
procedure CTX_REPORT.TOKEN_INFO(
    index_name      IN VARCHAR2,
    report          IN OUT NOCOPY CLOB,
    token           IN VARCHAR2,
    token_type      IN NUMBER,
    part_name       IN VARCHAR2 DEFAULT NULL,
    raw_info        IN BOOLEAN  DEFAULT FALSE,
    decoded_info    IN BOOLEAN  DEFAULT TRUE,
    report_format   IN VARCHAR2 DEFAULT FMT_TEXT
);
```

```sql
function CTX_REPORT.TOKEN_INFO(
    index_name      IN VARCHAR2,
    token           IN VARCHAR2,
    token_type      IN NUMBER,
    part_name       IN VARCHAR2 DEFAULT NULL,
    raw_info        IN VARCHAR2 DEFAULT 'N',
    decoded_info    IN VARCHAR2 DEFAULT 'Y',
    report_format   IN VARCHAR2 DEFAULT FMT_TEXT
) return clob;
```

**index_name**
Specify the name of the index.

**report**
Specify the CLOB locator to which to write the report.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller’s responsibility to free this temporary CLOB as needed.
The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call token may be case-sensitive, depending on the passed-in token type.

**token**
Specify the token text.

**token_type**
Specify the token type. THEME, ZONE, ATTR, PATH, and PATH ATTR tokens are case-sensitive.

Everything else gets passed through the lexer, so if the index’s lexer is case-sensitive, the token input is case-sensitive.

**part_name**
Specify the name of the index partition.

If the index is a local partitioned index, then part_name must be provided. TOKEN_INFO will apply to just that index partition.

**raw_info**
Specify TRUE to include a hex dump of the index data. If raw_info is TRUE, the report will include a hex dump of the raw data in the token_info column.

**decoded_info**
Specify decode and include docid and offset data. If decoded_info is FALSE, CTX_REPORT will not attempt to decode the token information. This is useful when you just want a dump of data.

**report_format**
Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX_REPORT.FMT_TEXT or CTX_REPORT.FMT_XML.
This is a helper function which translates an English name into a numeric token type. This is suitable for use with token_info, or any other CTX API which takes in a token_type.

```sql
function token_type(
    index_name in varchar2,
    type_name  in varchar2
) return number;
```

<table>
<thead>
<tr>
<th>Input</th>
<th>Meaning</th>
<th>Type Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>Normal text token.</td>
<td>0</td>
</tr>
<tr>
<td>THEME</td>
<td>Theme token.</td>
<td>1</td>
</tr>
<tr>
<td>ZONE SEC</td>
<td>Zone token.</td>
<td>2</td>
</tr>
<tr>
<td>ORIGINAL</td>
<td>Original form token</td>
<td>3</td>
</tr>
<tr>
<td>ATTR TEXT</td>
<td>Text that occurs in attribute.</td>
<td>4</td>
</tr>
<tr>
<td>ATTR SEC</td>
<td>Attribute section.</td>
<td>5</td>
</tr>
</tbody>
</table>
For FIELD types, the index metadata needs to be read, so if you are going to be calling this a lot for such things, you might want to consider caching the values in local variables rather than calling token_type over and over again.

The constant types (0 - 9) also have constants in this package defined.

**Example**

```python
 typenum := ctx_report.token_type('myindex', 'field author text');
```
This chapter provides reference information for using the CTX_THES package to manage and browse thesauri. These thesaurus functions are based on the ISO-2788 and ANSI Z39.19 standards except where noted.

Knowing how information is stored in your thesaurus helps in writing queries with thesaurus operators. You can also use a thesaurus to extend the knowledge base, which is used for ABOUT queries in English and French and for generating document themes.

CTX_THES contains the following stored procedures and functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_PHRASE</td>
<td>Alters thesaurus phrase.</td>
</tr>
<tr>
<td>ALTER_THESAURUS</td>
<td>Renames or truncates a thesaurus.</td>
</tr>
<tr>
<td>BT</td>
<td>Returns all broader terms of a phrase.</td>
</tr>
<tr>
<td>BTG</td>
<td>Returns all broader terms generic of a phrase.</td>
</tr>
<tr>
<td>BTI</td>
<td>Returns all broader terms instance of a phrase.</td>
</tr>
<tr>
<td>BTP</td>
<td>Returns all broader terms partitive of a phrase.</td>
</tr>
<tr>
<td>CREATE_PHRASE</td>
<td>Adds a phrase to the specified thesaurus.</td>
</tr>
<tr>
<td>CREATE_RELATION</td>
<td>Creates a relation between two phrases.</td>
</tr>
<tr>
<td>CREATE_THESAURUS</td>
<td>Creates the specified thesaurus.</td>
</tr>
<tr>
<td>CREATE_TRANSLATION</td>
<td>Creates a new translation for a phrase.</td>
</tr>
<tr>
<td>DROP_PHRASE</td>
<td>Removes a phrase from thesaurus.</td>
</tr>
<tr>
<td>DROP_RELATION</td>
<td>Removes a relation between two phrases.</td>
</tr>
</tbody>
</table>
DROP_THESAURUS
Drops the specified thesaurus from the thesaurus tables.

DROP_TRANSLATION
Drops a translation for a phrase.

HAS_RELATION
Tests for the existence of a thesaurus relation.

NT
Returns all narrower terms of a phrase.

NTG
Returns all narrower terms generic of a phrase.

NTI
Returns all narrower terms instance of a phrase.

NTP
Returns all narrower terms partitive of a phrase.

OUTPUT_STYLE
Sets the output style for the expansion functions.

PT
Returns the preferred term of a phrase.

RT
Returns the related terms of a phrase.

SN
Returns scope note for phrase.

SYN
Returns the synonym terms of a phrase.

THES_TT
Returns all top terms for phrase.

TR
Returns the foreign equivalent of a phrase.

TRSYN
Returns the foreign equivalent of a phrase, synonyms of the phrase, and foreign equivalent of the synonyms.

TT
Returns the top term of a phrase.

UPDATE_TRANSLATION
Updates an existing translation.

See Also: Chapter 3, "CONTAINS Query Operators" for more information about the thesaurus operators.
**ALTER_PHRASE**

Alters an existing phrase in the thesaurus. Only CTXSYS or thesaurus owner can alter a phrase.

**Syntax**

```
CTX_THES.ALTER_PHRASE(tname      in varchar2,
phrase     in varchar2,
op         in varchar2,
operand    in varchar2 default null);
```

**tname**
Specify thesaurus name.

**phrase**
Specify phrase to alter.

**op**
Specify the alter operation as a string or symbol. You can specify one of the following operations with the `op` and `operand` pair:

<table>
<thead>
<tr>
<th>op</th>
<th>meaning</th>
<th>operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENAME or CTX_THES.OP_RENAME</td>
<td>Rename phrase. If the new phrase already exists in the thesaurus, this procedure raises an exception.</td>
<td>Specify new phrase. You can include qualifiers to change, add, or remove qualifiers from phrases.</td>
</tr>
<tr>
<td>PT or CTX_THES.OP_PT</td>
<td>Make phrase the preferred term. Existing preferred terms in the synonym ring becomes non-preferred synonym.</td>
<td>(none)</td>
</tr>
<tr>
<td>SN or CTX_THES.OP_SN</td>
<td>Change the scope note on the phrase.</td>
<td>Specify new scope note.</td>
</tr>
</tbody>
</table>

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

Correct misspelled word in thesaurus:
```plaintext
ctx_thes.alter_phrase('thes1', 'tee', 'rename', 'tea');
```

Remove qualifier from mercury (metal):
```plaintext
ctx_thes.alter_phrase('thes1', 'mercury (metal)', 'rename', 'mercury');
```

Add qualifier to mercury:
```plaintext
ctx_thes.alter_phrase('thes1', 'mercury', 'rename', 'mercury (planet)');
```

Make Kowalski the preferred term in its synonym ring:
```plaintext
ctx_thes.alter_phrase('thes1', 'Kowalski', 'pt');
```

Change scope note for view cameras:
```plaintext
ctx_thes.alter_phrase('thes1', 'view cameras', 'sn', 'Cameras with lens focusing');
```
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

```sql
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:

<table>
<thead>
<tr>
<th>op</th>
<th>Meaning</th>
<th>operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENAME</td>
<td>Rename thesaurus. Returns an error if the new name already exists.</td>
<td>Specify new thesaurus name.</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTX_THES.OP_RENAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRUNCATE</td>
<td>Truncate thesaurus.</td>
<td>None.</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTX_THES.OP_TRUNCATE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

**Examples**

Rename thesaurus THES1 to MEDICAL:

```sql
ctx_thes.alter_thesaurus('thes1', 'rename', 'medical');
```

or

```sql
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');
This function returns all broader terms of a phrase as recorded in the specified thesaurus.

**Syntax 1: Table Result**

```sql
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**

```sql
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:

```sql
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about `EXP_TAB`.

**phrase**
Specify phrase to lookup in thesaurus.

**lvl**
Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.

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

This function returns a string of broader terms in the form:

(bt1)|{bt2}||{bt3} ...

Example

String Result
Consider a thesaurus named MY.THES that has an entry for cat as follows:

cat
  BT1 feline
  BT2 mammal
  BT3 vertebrate
  BT4 animal

To look up the broader terms for cat up to two levels, issue the following statements:

*set serveroutput on*

declare
terms varchar2(2000);
begin
  terms := ctx_thes.bt('CAT', 2, 'MY_THES');
dbms_output.put_line('The broader expansion for CAT is: '||terms);
end;

This code produces the following output:
The broader expansion for CAT is: {cat}|{feline}|{mammal}

Table Result
The following code does a broader term lookup for white wolf using the table result:

*set serveroutput on*

declare
  xtab ctx_thes.exp_tab;
begin
  ctx_thes.bt(xtab, 'white wolf', 2, 'my_thesaurus');
  for i in 1..xtab.count loop
dbms_output.put_line(xtab(i).rel||' '||xtab(i).phrase);
  end loop;
end;
end loop;
end;

This code produces the following output:

PHRASE WHITE WOLF
BT WOLF
BT CANINE
BT ANIMAL

Related Topics

OUTPUT_STYLE

Broader Term (BT, BTG, BTP, BTI) Operators in Chapter 3, “CONTAINS Query Operators”
This function returns all broader terms generic of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```sql
CTX_THESES.BTG(restab IN OUT NOCOPY EXP_TAB,
    phrase IN VARCHAR2,
    lvl IN NUMBER DEFAULT 1,
    tname IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```sql
CTX_THESES.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:

```sql
type exp_rec is record (  
xrel varchar2(12),  
xlevel number,  
xphrase varchar2(256)
);

type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX_THESES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about EXP_TAB.

**phrase**
Specify phrase to lookup in thesaurus.

**lvl**
Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.

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

This function returns a string of broader terms generic in the form:

\{bt1\}|\{bt2\}|\{bt3\} ...

Example

To look up the broader terms generic for cat up to two levels, issue the following statements:

```
set serveroutput on
declare
    terms varchar2(2000);
begin
    terms := ctx_thes.btg('CAT', 2, 'MY_THES');
    dbms_output.put_line('the broader expansion for CAT is: '||terms);
end;
```

Related Topics

- OUTPUT_STYLE

Broader Term (BT, BTG, BTP, BTI) Operators in Chapter 3, "CONTAINS Query Operators"
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:

```plaintext
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about EXP_TAB.

**phrase**
Specify phrase to lookup in thesaurus.

**lvl**
Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.

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

This function returns a string of broader terms instance in the form:

\{bt1\}|\{bt2\}|\{bt3\} ...

Example

To look up the broader terms instance for *cat* up to two levels, issue the following statements:

```sql
set serveroutput on
declare
    terms varchar2(2000);
begin
    terms := ctx_thes.bti('CAT', 2, 'MY_THES');
    dbms_output.put_line('the broader expansion for CAT is: '||terms);
end;
```

Related Topics

- OUTPUT_STYLE
- Broader Term (BT, BTG, BTP, BTI) Operators in Chapter 3, "CONTAINS Query Operators"
This function returns all broader terms partitive of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```
CTX_THES.BTP(restab IN OUT NOCOPY EXP_TAB,
    phrase IN VARCHAR2,
    lvl IN NUMBER DEFAULT 1,
    tname IN VARCHAR2 DEFAULT 'DEFAULT');
```

Syntax 2: String Result

```
CTX_THES.BTP(phrase IN VARCHAR2,
    lvl IN NUMBER DEFAULT 1,
    tname IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

**restab**
Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
```

```
type exp_tab is table of exp_rec index by binary_integer;
```

See Also: "CTX_THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about EXP_TAB.

**phrase**
Specify phrase to lookup in thesaurus.

**lvl**
Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.

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

This function returns a string of broader terms in the form:

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

Example

To look up the 2 broader terms partitive for cat, issue the following statements:

```
declare
terms  varchar2(2000);
begin
  terms := ctx_thes.btp('CAT', 2, 'MY_THES');
  dbms_output.put_line('the broader expansion for CAT is: '||terms);
end;
```

Related Topics

OUTPUT_STYLE

Broader Term (BT, BTG, BTP, BTI) Operators in Chapter 3, "CONTAINS Query Operators"
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

```sql
CTX_THES.CREATE_PHRASE(tname IN VARCHAR2,
                         phrase IN VARCHAR2,
                         rel IN VARCHAR2 DEFAULT NULL,
                         relname IN VARCHAR2 DEFAULT NULL);
```

- **tname**
  Specify the name of the thesaurus in which the new phrase is added or the existing phrase is located.

- **phrase**
  Specify the phrase to be added to a thesaurus or the phrase for which a new relationship is created.

- **rel**
  Specify the new relationship between `phrase` and `relname`. This parameter is supported only for backward compatibility. Use CTX_THES.CREATE_RELATION to create new relations in a thesaurus.

- **relname**
  Specify the existing phrase that is related to `phrase`. This parameter is supported only for backward compatibility. Use CTX_THES.CREATE_RELATION to create new relations in a thesaurus.

### Returns

The ID for the entry.
Examples

Creating Entries for Phrases
In this example, two new phrases (*os* and *operating system*) are created in a thesaurus named `tech_thes`.

```java
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**

```sql
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:

<table>
<thead>
<tr>
<th>relation</th>
<th>meaning</th>
<th>relphrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT*/NT*</td>
<td>Add hierarchical relation. Specify related phrase. The relationship is interpreted from phrase to relphrase.</td>
<td></td>
</tr>
<tr>
<td>RT</td>
<td>Add associative relation. Specify phrase to associate.</td>
<td></td>
</tr>
<tr>
<td>SYN</td>
<td>Add phrase to a synonym ring. Specify an existing phrase in the synonym ring.</td>
<td></td>
</tr>
<tr>
<td>Specify language</td>
<td>Add translation for a phrase.</td>
<td>Specify new translation phrase.</td>
</tr>
</tbody>
</table>
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

```sql
CTX_THES.CREATE_THESAURUS (name IN VARCHAR2,
casesens IN BOOLEAN DEFAULT FALSE);
```

**name**
Specify the name of the thesaurus to be created. The name of the thesaurus must be unique. If a thesaurus with the specified name already exists, CREATE_THESAURUS returns an error and does not create the thesaurus.

**casesens**
Specify whether the thesaurus to be created is case-sensitive. If casesens is `true`, Oracle Text retains the cases of all terms entered in the specified thesaurus. As a result, queries that use the thesaurus are case-sensitive.

Example

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

Use this procedure to create a new translation for a phrase in a specified language.

Syntax

```sql
CTX_THES.CREATE_TRANSLATION(tname in varchar2,
                               phrase in varchar2,
                               language in varchar2,
                               translation in varchar2);
```

tname
Specify the name of the thesaurus, using no more than 30 characters.

phrase
Specify the phrase in the thesaurus to which to add a translation. Phrase must already exist in the thesaurus, or an error is raised.

language
Specify the language of the translation, using no more than 10 characters.

translation
Specify the translated term, using no more than 256 characters.

If a translation for this phrase already exists, this new translation is added without removing that original translation, so long as that original translation is not the same. Adding the same translation twice results in an error.

Example

The following code adds the Spanish translation for dog to my_thes:

```sql
begin
    ctx_thes.create_translation('my_thes', 'dog', 'SPANISH', 'PERRO');
end;
```
Removes a phrase from the thesaurus. Only thesaurus owner and CTXSYS can invoke this procedure on a given thesaurus.

**Syntax**

```
CTX_THES.DROP_PHRASE(tname in varchar2,
                     phrase in varchar2);
```

**tname**
Specify thesaurus name.

**phrase**
Specify phrase to drop. If phrase is a disambiguated homograph, you must include the qualifier. When phrase does not exist in tname, this procedure raises and exception.

BT* / NT* relations are patched around the dropped phrase. For example, if A has a BT B, and B has BT C, after B is dropped, A has BT C.

When a word has multiple broader terms, then a relationship is established for each narrower term to each broader term.

Note that BT, BTG, BTP, and BTI are separate hierarchies, so if A has BTG B, and B has BTI C, when B is dropped, there is no relation implicitly created between A and C.

RT relations are not patched. For example, if A has RT B, and B has RT C, then if B is dropped, there is no associative relation created between A and C.

**Example**

Assume you have the following relations defined in *mythes*:

```
wolf
  BT canine
canine
  BT animal
```

You drop phrase *canine*:

```
begin
  ctx_thes.drop_phrase('mythes', 'canine');
end;
```
The resulting thesaurus is patched and looks like:

wolf
  BT animal
DROP_RELATION

Removes a relation between two phrases from the thesaurus.

Note: CTX_THES.DROP_RELATION removes only the relation between two phrases. Phrases are never removed by this call.

Only thesaurus owner and CTXSYS can invoke this procedure on a given thesaurus.

Syntax

```
CTX_THES.DROP_RELATION(tname    in    varchar2,
  phrase    in    varchar2,
  rel       in    varchar2,
  relphrase in    varchar2 default null);
```

tname
Specify thesaurus name.

phrase
Specify the filing phrase.

rel
Specify relation to drop. The relation is from phrase to relphrase. You can specify one of the following relations:

<table>
<thead>
<tr>
<th>relation</th>
<th>meaning</th>
<th>relphrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT*/NT*</td>
<td>Remove hierarchical relation.</td>
<td>Optional specify relphrase. If not provided, all relations of that type for the phrase are removed.</td>
</tr>
<tr>
<td>RT</td>
<td>Remove associative relation.</td>
<td>Optionally specify relphrase. If not provided, all RT relations for the phrase are removed.</td>
</tr>
<tr>
<td>SYN</td>
<td>Remove phrase from its synonym ring.</td>
<td>(none)</td>
</tr>
</tbody>
</table>
**DROP_RELATION**

<table>
<thead>
<tr>
<th>relation</th>
<th>meaning</th>
<th>relphrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT</td>
<td>Remove preferred term designation from the phrase. The phrase remains in the synonym ring.</td>
<td>(none)</td>
</tr>
<tr>
<td>language</td>
<td>Remove a translation from a phrase.</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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:

```plaintext
dog
  BT animal
```

To remove this relation, specify the arguments as follows:

```plaintext
begin
  CTX_THES.DROP_RELATION('thes', 'dog', 'BT', 'animal');
end;
```

You can also remove this relation using NT as follows:

```plaintext
begin
  CTX_THES.DROP_RELATION('thes', 'animal', 'NT', 'dog');
end;
```

**Example**

Remove relation VEHICLE NT CAR:

```plaintext
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')
The DROP_THESAURUS procedure deletes the specified thesaurus and all of its entries from the thesaurus tables.

Syntax

```sql
CTX_THES.DROP_THESAURUS(name IN VARCHAR2);
```

**name**
Specify the name of the thesaurus to be dropped.

Examples

```sql
begin
ctx_thes.drop_thesaurus('tech_thes');
end;
```
Use this procedure to remove one or more translations for a phrase.

Syntax

```
CTX_THES.DROP_TRANSLATION (tname in varchar2,
phrase in varchar2,
language in varchar2 default null,
translation in varchar2 default null);
```

**tname**
Specify the name of the thesaurus, using no more than 30 characters.

**phrase**
Specify the phrase in the thesaurus to which to remove a translation. The phrase must already exist in the thesaurus or an error is raised.

**language**
Optionally, specify the language of the translation, using no more than 10 characters. If not specified, the translation must also not be specified and all translations in all languages for the phrase are removed. An error is raised if the phrase has no translations.

**translation**
Optionally, specify the translated term to remove, using no more than 256 characters. If no such translation exists, an error is raised.

**Example**

The following code removes the Spanish translation for *dog*:

```
begin
   ctx_thes.drop_translation('my_thes', 'dog', 'SPANISH', 'PERRO');
end;
```

To remove all translations for *dog* in all languages:

```
begin
   ctx_thes.drop_translation('my_thes', 'dog');
end;
```
HAS_RELATION

Use this procedure to test that a thesaurus relation exists without actually doing the expansion. The function returns TRUE if the phrase has any of the relations in the specified list.

Syntax

```
CTX_THES.HAS_RELATION(phrase in varchar2,
   rel in varchar2,
   tname in varchar2 default 'DEFAULT')
returns boolean;
```

**phrase**
Specify the phrase.

**rel**
Specify a single thesaural relation or a comma-delimited list of relations, except PT. Specify 'ANY' for any relation.

**tname**
Specify the thesaurus name.

Example

The following example returns TRUE if the phrase cat in the DEFAULT thesaurus has any broader terms or broader generic terms:

```
set serveroutput on
result boolean;
begin
   result := ctx_thes.has_relation('cat','BT,BTG');
   if (result) then dbms_output.put_line('TRUE');
   else dbms_output.put_line('FALSE');
   end if;
end;
```
This function returns all narrower terms of a phrase as recorded in the specified thesaurus.

**Syntax 1: Table Result**

```
CTX_THES.NT(restab IN OUT NOCOPY EXP_TAB,
    phrase IN VARCHAR2,
    lvl IN NUMBER DEFAULT 1,
    tname IN VARCHAR2 DEFAULT 'DEFAULT');
```

**Syntax 2: String Result**

```
CTX_THES.NT(phrase IN VARCHAR2,
    lvl IN NUMBER DEFAULT 1,
    tname IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

**restab**
Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);

type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about EXP_TAB.

**phrase**
Specify phrase to lookup in thesaurus.

**lvl**
Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

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

This function returns a string of narrower terms in the form:

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

Example

String Result

Consider a thesaurus named MY_THES that has an entry for cat as follows:

cat
NT domestic cat
NT wild cat
BT mammal
mammal
BT animal
domestic cat
NT Persian cat
NT Siamese cat

to look up the narrower terms for cat down to two levels, issue the following statements:

declare
terms varchar2(2000);
begin
 terms := ctx_thes.nt('CAT', 2, 'MY_THES');
dbms_output.put_line('the narrower expansion for CAT is: '||terms);
end;

This code produces the following output:

the narrower expansion for CAT is: {cat}|{domestic cat}|{Persian cat}|{Siamese cat}| {wild cat}

Table Result

The following code does a 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) ||


This code produces the following output:

PHRASE CANINE
NT WOLF (Canis lupus)
NT WHITE WOLF
NT GREY WOLF
NT DOG (Canis familiaris)
NT PIT BULL
NT DASCHUND
NT CHIHUAHUA
NT HYENA (Canis mesomelas)
NT COYOTE (Canis latrans)

Related Topics

OUTPUT_STYLE

Narrower Term (NT, NTG, NTP, NTI) Operators in Chapter 3, "CONTAINS Query Operators"
This function returns all narrower terms generic of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```sql
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

```sql
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:

```sql
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
```

```sql
type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about `EXP_TAB`.

**phrase**
Specify phrase to lookup in thesaurus.

**lvl**
Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

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

This function returns a string of narrower terms generic in the form:

\{(nt1)\}|\{(nt2)\}|\{(nt3)\} ...

Example

To look up the narrower terms generic for cat down to two levels, issue the following statements:

```sql
declare
terms varchar2(2000);
begin
terms := ctx_thes.ntg('CAT', 2, 'MY_THES');
dbms_output.put_line('the narrower expansion for CAT is: '||terms);
end;
```

Related Topics

OUTPUT_STYLE

Narrower Term (NT, NTG, NTP, NTI) Operators in Chapter 3, "CONTAINS Query Operators"
This function returns all narrower terms instance of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

```sql
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

```sql
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:

```sql
type exp_rec is record (  
    xrel varchar2(12),  
    xlevel number,  
    xphrase varchar2(256)  
);  
type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about `EXP_TAB`.

**phrase**
Specify phrase to lookup in thesaurus.

**lvl**
Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

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

This function returns a string of narrower terms instance in the form:

(nt1)||nt2)||nt3) ...

Example

To look up the narrower terms instance for cat down to two levels, issue the following statements:

```sql
declare
terms varchar2(2000);
begin
terms := ctx_thes.nti('CAT', 2, 'MY_THES');
dbms_output.put_line('the narrower expansion for CAT is: '||terms);
end;
```

Related Topics

OUTPUT_STYLE

Narrower Term (NT, NTG, NTP, NTI) Operators in Chapter 3, "CONTAINS Query Operators"
This function returns all narrower terms partitive of a phrase as recorded in the specified thesaurus.

**Syntax 1: Table Result**

```sql
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**

```sql
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:

```sql
type exp_rec is record (xrel varchar2(12), xlevel number, xphrase varchar2(256));
type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about `EXP_TAB`.

**phrase**
Specify phrase to lookup in thesaurus.

**lvl**
Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

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

This function returns a string of narrower terms partitive in the form:

(nt1)|{nt2}|{nt3} ...

Example

To look up the narrower terms partitive for cat down to two levels, issue the following statements:

```
declare
    terms varchar2(2000);
begin
    terms := ctx_thes.ntp('CAT', 2, 'MY_THES');
    dbms_output.put_line('the narrower expansion for CAT is: '||terms);
end;
```

Related Topics

OUTPUT_STYLE

Narrower Term (NT, NTG, NTP, NTI) Operators in Chapter 3, "CONTAINS Query Operators"
OUTPUT_STYLE

Sets the output style for the return string of the CTX_THES expansion functions. This procedure has no effect on the table results to the CTX_THES expansion functions.

Syntax

```plaintext
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
```
This function returns the preferred term of a phrase as recorded in the specified thesaurus.

**Syntax 1: Table Result**

```sql
CTX.THES.PT(restab IN OUT NOCOPY EXP_TAB,
            phrase IN VARCHAR2,
            tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN varchar2;
```

**Syntax 2: String Result**

```sql
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:

```sql
type exp_rec is record (xrel varchar2(12), xlevel number, xphrase varchar2(256));
type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX.THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about EXP_TAB.

**phrase**
Specify phrase to lookup in thesaurus.

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

**Returns**
This function returns the preferred term as a string in the form:

```
{pt}
```
Example

Consider a thesaurus MY_THES with the following preferred term definition for automobile:

AUTOMOBILE
  PT CAR

To look up the preferred term for automobile, execute the following code:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.pt('AUTOMOBILE', 'MY_THES');
  dbms_output.put_line('The preferred term for automobile is: ' || terms);
end;
```

Related Topics

OUTPUT_STYLE
Preferred Term (PT) Operator in Chapter 3, "CONTAINS Query Operators"
This function returns the related terms of a term in the specified thesaurus.

Syntax 1: Table Result

    CTX_THES.RT(restab IN OUT NOCOPY EXP_TAB,
                phrase IN VARCHAR2,
                tname  IN VARCHAR2 DEFAULT 'DEFAULT');

Syntax 2: String Result

    CTX_THES.RT(phrase IN VARCHAR2,
                tname  IN VARCHAR2 DEFAULT 'DEFAULT')
    RETURN varchar2;

    restab
    Optionally, specify the name of the expansion table to store the results. This table
    must be of type EXP_TAB which the system defines as follows:

    type exp_rec is record (              
        xrel varchar2(12),
        xlevel number,              
        xphrase varchar2(256)       
    );
    type exp_tab is table of exp_rec index by binary_integer;

    See Also: "CTX_THES Result Tables and Data Types" in
              Appendix A, "Result Tables" for more information about EXP_TAB.

    phrase
    Specify phrase to lookup in thesaurus.

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

Returns

This function returns a string of related terms in the form:

    (rt1)||(rt2)||(/rt3)|| ...
Example

Consider a thesaurus MY_THES with the following related term definition for dog:

```
DOG
  RT WOLF
  RT HYENA
```

To look up the related terms for dog, execute the following code:

```sql
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 code produces the following output:

```
The related terms for dog are: {dog}|{wolf}|{hyena}
```

Related Topics

- OUTPUTSTYLE

Related Term (RT) Operator in Chapter 3, "CONTAINS Query Operators"
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
This function returns all synonyms of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

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

Syntax 2: String Result

```sql
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:

```sql
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

See Also: "CTX_THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about EXP_TAB.

phrase
Specify phrase to lookup in thesaurus.

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

Returns
This function returns a string of the form:

```
{syn1}|{syn2}|{syn3} ... 
```
Example

String Result
Consider a thesaurus named ANIMALS that has an entry for cat as follows:

```
CAT
  SYN KITTY
  SYN FELINE
```
To look-up the synonym for cat and obtain the result as a string, issue the following statements:

```
declare
  synonyms varchar2(2000);
begn
  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;
begn
  ctx_thes.syn(xtab, 'canine', 'my_thesaurus');
  for i in 1..xtab.count loop
      dbms_output.put_line(lpad(' ', 2*xtab(i).xlevel) ||
                      xtab(i).xrel || ' ' || xtab(i).xphrase);
  end loop;
end;
```
This code produces the following output:
```
PHRASE CANINE
PT DOG
SYN PUPPY
SYN MUTT
SYN MONGREL
```
Related Topics

OUTPUT_STYLE
SYNonym (SYN) Operator in Chapter 3, "CONTAINS Query Operators"
This procedure finds and returns all top terms of a thesaurus. A top term is defined as any term which has a narrower term but has no broader terms.

This procedure differs from TT in that TT takes in a phrase and finds the top term for that phrase, but THES_TT searches the whole thesaurus and finds all top terms.

Large Thesauri

Since this procedure searches the whole thesaurus, it can take some time on large thesauri. Oracle recommends that you not call this often for such thesauri. Instead, your application should call this once, store the results in a separate table, and use those stored results.

Syntax

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

**restab**
Specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, 'Result Tables' for more information about EXP_TAB.

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

Returns

This procedure returns all top terms and stores them in restab.
For a given mono-lingual thesaurus, this function returns the foreign language equivalent of a phrase as recorded in the thesaurus.

**Note:** Foreign language translation is not part of the ISO-2788 or ANSI Z39.19 thesaural standards. The behavior of TR is specific to Oracle Text.

### Syntax 1: Table Result

```sql
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

```sql
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:

```sql
type exp_rec is record (    xrel varchar2(12),    xlevel number,    xphrase varchar2(256)    );
type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about `EXP_TAB`.

**phrase**

Specify phrase to lookup in thesaurus.
lang
Specify the foreign language. Specify 'ALL' for all translations of phrase.

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

Returns
This function returns a string of foreign terms in the form:
{ft1}|{ft2}|{ft3} ...

Example
Consider a thesaurus MY_THES with the following entries for \textit{cat}:

\begin{verbatim}
cat
  SPANISH: gato
  FRENCH:  chat
  SYN lion
    SPANISH: leon
\end{verbatim}

To look up the translation for \textit{cat}, you can issue the following statements:

\begin{verbatim}
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;
\end{verbatim}

This codes produces the following output:

the translations for CAT are: {CAT}|{CHAT}|{GATO}
the Spanish translations for CAT are: {CAT}|{GATO}

Related Topics
OUTPUT_STYLE
Translation Term (TR) Operator in Chapter 3, "CONTAINS Query Operators"
For a given mono-lingual thesaurus, this function returns the foreign equivalent of a phrase, synonyms of the phrase, and foreign equivalent of the synonyms as recorded in the specified thesaurus.

**Note:** Foreign language translation is not part of the ISO-2788 or ANSI Z39.19 thesaural standards. The behavior of TRSYN is specific to Oracle Text.

**Syntax 1: Table Result**

```sql
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**

```sql
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:

```sql
type exp_rec is record (  
    xrel varchar2(12),  
    xlevel number,  
    xphrase varchar2(256)  
);  
type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about EXP_TAB.

**phrase**
Specify phrase to lookup in thesaurus.
**lang**
Specify the foreign language. Specify 'ALL' for all translations of *phrase*.

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

**Returns**
This function returns a string of foreign terms in the form:

```
{ft1}|{ft2}|{ft3} ...
```

**Example**
Consider a thesaurus **MY_THES** with the following entries for *cat*:

```
cat
    SPANISH: gato
    FRENCH: chat
    SYN lion
        SPANISH: leon
```

To look up the translation and synonyms for *cat*, you can issue the following statements:

```sql
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 (TRSYSN) Operator in Chapter 3, "CONTAINS Query Operators"
This function returns the top term of a phrase as recorded in the specified thesaurus.

**Syntax 1: Table Result**

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

**Syntax 2: String Result**

```
CTX_THES.TT(phrase IN VARCHAR2,
tname  IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN varchar2;
```

**restab**
Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
  xrel varchar2(12),
  xlevel number,
  xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

**See Also:** "CTX_THES Result Tables and Data Types" in Appendix A, "Result Tables" for more information about EXP_TAB.

**phrase**
Specify phrase to lookup in thesaurus.

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

**Returns**
This function returns the top term string in the form:

```
{tt}
```
Example

Consider a thesaurus MY_THES with the following broader term entries for dog:

DOG
  BT1 CANINE
  BT2 MAMMAL
  BT3 VERTEBRATE
  BT4 ANIMAL

To look up the top term for DOG, execute the following code:

declare
  terms varchar2(2000);
begin
  terms := ctx_thes.tt('DOG','MY_THES');
  dbms_output.put_line('The top term for DOG is: '||terms);
end;

This code produces the following output:

The top term for dog is: {ANIMAL}

Related Topics

OUTPUT_STYLE

Top Term (TT) Operator in Chapter 3, "CONTAINS Query Operators"
UPDATE_TRANSLATION

Use this procedure to update an existing translation.

Syntax

```sql
CTX_THES.UPDATE_TRANSLATION(tname in varchar2,
                             phrase in varchar2,
                             language in varchar2,
                             translation in varchar2,
                             new_translation in varchar2);
```

tname
Specify the name of the thesaurus, using no more than 30 characters.

phrase
Specify the phrase in the thesaurus to which to update a translation. The phrase
must already exist in the thesaurus or an error is raised.

language
Specify the language of the translation, using no more than 10 characters.

translation
Specify the translated term to update. If no such translation exists, an error is raised.
You can specify NULL if there is only one translation for the phrase. An error is raised
if there is more than one translation for the term in the specified language.

new_translation
Optionally, specify the new form of the translated term.

Example

The following code updates the Spanish translation for dog:

```sql
begin
  ctx_thes.update_translation('my_thes', 'dog', 'SPANISH:', 'PERRO', 'CAN');
end;
```
This chapter provides reference information for using the CTX_ULEXER PL/SQL package to use with the user-lexer.

CTX_ULEXER declares the following type:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WILDCARD_TAB</td>
<td>Index-by table type you use to specify the offset of characters to be treated as wildcard characters by the user-defined lexer query procedure.</td>
</tr>
</tbody>
</table>
TYPE WILDCARD_TAB IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;

Use this index-by table type to specify the offset of those characters in the query word to be treated as wildcard characters by the user-defined lexer query procedure.
This chapter discusses the executables shipped with Oracle Text. The following topics are discussed:

- **Thesaurus Loader (ctxload)**
- **Knowledge Base Extension Compiler (ctxkbtc)**
- **Lexical Compiler (ctxlc)**

**Thesaurus Loader (ctxload)**

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

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

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

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

**Text Loading**

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

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

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

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

Mandatory Arguments

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

The user name and password can be followed immediately by @sqlnet_address to
permit logon to remote databases. The value for sqlnet_address is a database connect
string. If the TWO_TASK environment variable is set to a remote database, you do
not have to specify a value for sqlnet_address to connect to the database.

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

You use object_name to identify the thesaurus in queries that use thesaurus
operators.

Note: Thesaurus name must be unique. If the name specified for
the thesaurus is identical to an existing thesaurus, ctxload returns
an error and does not overwrite the existing thesaurus.

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

-file file_name
When ctxload is used to import a thesaurus, use file_name to specify the name of
the import file which contains the thesaurus entries.
When `ctxload` is used to export a thesaurus, use `file_name` to specify the name of the export file created by `ctxload`.

---

**Note:** If the name specified for the thesaurus dump file is identical to an existing file, `ctxload` overwrites the existing file.

---

### Optional Arguments

**-thes**
Import a thesaurus. Specify the source file with the `-file` argument. You specify the name of the thesaurus to be imported with `-name`.

**-thescase y | n**
Specify `y` to create a case-sensitive thesaurus with the name specified by `-name` and populate the thesaurus with entries from the thesaurus import file specified by `-file`. If `-thescase` is `y` (the thesaurus is case-sensitive), `ctxload` enters the terms in the thesaurus exactly as they appear in the import file.

The default for `-thescase` is `n` (case-insensitive thesaurus)

---

**Note:** `-thescase` is valid for use with only the `-thes` argument.

---

**-thesdump**
Export a thesaurus. Specify the name of the thesaurus to be exported with the `-name` argument. Specify the destination file with the `-file` argument.

**-log**
Specify the name of the log file to which `ctxload` writes any national-language supported (Globalization Support) messages generated during processing. If you do not specify a log file name, the messages appear on the standard output.

**-trace**
Enables SQL statement tracing using `ALTER SESSION SET SQL_TRACE TRUE`. This command captures all processed SQL statements in a trace file, which can be used for debugging. The location of the trace file is operating-system dependent and can be modified using the `USER_DUMP_DEST` initialization parameter.

**See Also:** For more information about SQL trace and the `USER_DUMP_DEST` initialization parameter, see *Oracle Database Administrator’s Guide*
-pk
Specify the primary key value of the row to be updated or exported.
When the primary key is compound, you must enclose the values within double quotes and separate the keys with a comma.

-export
Exports the contents of a CLOB or BLOB column in a database table into the operating system file specified by -file. ctxload exports the CLOB or BLOB column in the row specified by -pk.
When you use the -export, you must specify a primary key with -pk.

-update
Updates the contents of a CLOB or BLOB column in a database table with the contents of the operating system file specified by -file. ctxload updates the CLOB or BLOB column in for the row specified by -pk.
When you use -update, you must specify a primary key with -pk.

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

See Also: For more document loading examples, see Appendix C, "Loading Examples".

Thesaurus Import Example
The following example imports a thesaurus named tech_doc from an import file named tech_thesaurus.txt:
ctxload -user jsmith/123abc -thes -name tech_doc -file tech_thesaurus.txt

Thesaurus Export Example
The following example dumps the contents of a thesaurus named tech_doc into a file named tech_thesaurus.out:
ctxload -user jsmith/123abc -thesdump -name tech_doc -file tech_thesaurus.out
Knowledge Base Extension Compiler (ctxkbtc)

The knowledge base is the information source Oracle Text uses to perform theme analysis, such as theme indexing, processing ABOUT queries, and document theme extraction with the CTX_DOC package. A knowledge base is supplied for English and French.

With the ctxkbtc compiler, you can do the following:

- Extend your knowledge base by compiling one or more thesauri with the Oracle Text knowledge base. The extended information can be application-specific terms and relationships. During theme analysis, the extended portion of the knowledge base overrides any terms and relationships in the knowledge base where there is overlap.

- Create a new user-defined knowledge base by compiling one or more thesauri. In languages other than English and French, this feature can be used to create a language-specific knowledge base.

---

**Note:** Only CTXSYS can extend the knowledge base.

---

**See Also:** For more information about the knowledge base packaged with Oracle Text, see http://otn.oracle.com/products/text/

For more information about the ABOUT operator, see ABOUT operator in Chapter 3, "CONTAINS Query Operators".

For more information about document services, see Chapter 8, "CTX_DOC Package".

Knowledge Base Character Set

Knowledge bases can be in any single-byte character set. Supplied knowledge bases are in WE8ISO8859P1. You can store an extended knowledge base in another character set such as US7ASCII.

**ctxkbtc Syntax**

```
ctxkbtc -user uname/passwd
[-name thesname1 [thesname2 ... thesname16]]
[-revert]
[-stoplist stoplistname]
```
Knowledge Base Extension Compiler (ctxkbtc)

[-verbose]
[-log filename]

-user
Specify the user name and password for the administrator creating an extended knowledge base. This user must have write permission to the ORACLE_HOME directory.

-name thesname1 [thesname2 ... thesname16]
Specify the name(s) of the thesauri (up to 16) to be compiled with the knowledge base to create the extended knowledge base. The thesauri you specify must already be loaded with ctxload with the "-thescase Y" option

-revert
Reverts the extended knowledge base to the default knowledge base provided by Oracle Text.

-stoplist stoplistname
Specify the name of the stoplist. Stopwords in the stoplist are added to the knowledge base as useless words that are prevented from becoming themes or contributing to themes. You can still add stopthemes after running this command using CTX_DLL.ADD_STOPTHEME.

-verbose
Displays all warnings and messages, including non-Globalization Support messages, to the standard output.

-log
Specify the log file for storing all messages. When you specify a log file, no messages are reported to standard out.

ctxkbtc Usage Notes

- Before running ctxkbtc, you must set the NLS_LANG environment variable to match the database character set.
- The user issuing ctxkbtc must have write permission to the ORACLE_HOME, since the program writes files to this directory.
- Before being compiled, each thesaurus must be loaded into Oracle Text case sensitive with the "-thescase Y" option in ctxload.
- Running ctxkbtc twice removes the previous extension.
ctxkbtc Limitations

The ctxkbtc program has the following limitations:

- When upgrading or downgrading your database to a different release, Oracle recommends that you recompile your extended knowledge base in the new environment for theme indexing and related features to work correctly.
- Knowledge base extension cannot be performed when theme indexing is being performed. In addition, any SQL sessions that are using Oracle Text functions must be exited and reopened to make use of the extended knowledge base.
- There can be only one user extension for each language for each installation. Since a user extension affects all users at the installation, only the CTXSYS user can extend the knowledge base.

ctxkbtc Constraints on Thesaurus Terms

Terms are case sensitive. If a thesaurus has a term in uppercase, for example, the same term present in lowercase form in a document will not be recognized.

The maximum length of a term is 80 characters.

Disambiguated homographs are not supported.

ctxkbtc Constraints on Thesaurus Relations

The following constraints apply to thesaurus relations:

- BTG and BTP are the same as BT. NTG and NTP are the same as NT.
- Only preferred terms can have a BT, NTs or RTs.
- If a term has no USE relation, it will be treated as its own preferred term.
- If a set of terms are related by SYN relations, only one of them may be a preferred term.
- An existing category cannot be made a top term.
- There can be no cycles in BT and NT relations.
- A term can have at most one preferred term and at most one BT. A term may have any number of NTs.
- An RT of a term cannot be an ancestor or descendant 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 preceding rules. For example, if a term has multiple BTs, it ignores all but the last one it encounters.

Similarly, BTs between existing knowledge base categories will only result in a warning message.

Such violations are not recommended since they might produce undesired results.

---

**Extending the Knowledge Base**

You can extend the supplied knowledge base by compiling one or more thesauri with the Oracle Text knowledge base. The extended information can be application-specific terms and relationships. During theme analysis, the extended portion of the knowledge base overrides any terms and relationships in the knowledge base where there is overlap.

When extending the knowledge base, Oracle recommends that new terms be linked to one of the categories in the knowledge base for best results in theme proving when appropriate.

**See Also:** For complete description of the supplied knowledge base, see http://otn.oracle.com/products/text/

If new terms are kept completely disjoint from existing categories, fewer themes from new terms will be proven. The result of this is poorer precision and recall with `ABOUT` queries as well poor quality of gists and theme highlighting.

You link new terms to existing terms by making an existing term the broader term for the new terms.

**Example for Extending the Knowledge Base**

You purchase a medical thesaurus `medthes` containing a hierarchy of medical terms. The four top terms in the thesaurus are the following:

- Anesthesia and Analgesia
- Anti-Allergic and Respiratory System Agents
- Anti-Inflammatory Agents, Antirheumatic Agents, and Inflammation Mediators
- Antineoplastic and Immunosuppressive Agents

To link these terms to the existing knowledge base, add the following entries to the medical thesaurus to map the new terms to the existing health and medicine branch:

```
health and medicine
NT Anesthesia and Analgesia
NT Anti-Allergic and Respiratory System Agents
NT Anti-Inflammatory Agents, Antirheumatic Agents, and Inflammation Mediators
NT Antineoplastic and Immunosuppressive Agents
```

Set your Globalization Support language environment variable to match the database character set. For example, if your database character set is WE8ISO8859P1 and you are using American English, set your NLS_LANG as follows:

```
setenv NLS_LANG AMERICAN_AMERICA.WE8ISO8859P1
```

Assuming the medical thesaurus is in a file called med.thes, you load the thesaurus as medthes with ctxload as follows:

```
ctxload -thes -thescase y -name medthes -file med.thes -user ctxsys/ctxsys
```

To link the loaded thesaurus medthes to the knowledge base, use ctxkbtc as follows:

```
cctxkbtc -user ctxsys/ctxsys -name medthes
```

### Adding a Language-Specific Knowledge Base

You can extend theme functionality to languages other than English or French by loading your own knowledge base for any single-byte whitespace delimited language, including Spanish.

Theme functionality includes theme indexing, ABOUT queries, theme highlighting, and the generation of themes, gists, and theme summaries with the CTX_DOC PL/SQL package.

You extend theme functionality by adding a user-defined knowledge base. For example, you can create a Spanish knowledge base from a Spanish thesaurus.

To load your language-specific knowledge base, follow these steps:

1. Load your custom thesaurus using ctxload.
2. Set NLS_LANG so that the language portion is the target language. The charset portion must be a single-byte character set.

3. Compile the loaded thesaurus using `ctxkbtc`:

   ```
   ctxkbtc -user ctxsys/ctxsys -name my_lang_thes
   ```

   This command compiles your language-specific knowledge base from the loaded thesaurus. To use this knowledge base for theme analysis during indexing and ABOUT queries, specify the NLS_LANG language as the THEME_LANGUAGE attribute value for the BASIC_LEXER preference.

Limitations for Adding a Knowledge Base

The following limitations hold for adding knowledge bases:

- Oracle Text supplies knowledge bases in English and French only. You must provide your own thesaurus for any other language.
- You can only add knowledge bases for languages with single-byte character sets. You cannot create a knowledge base for languages which can be expressed only in multibyte character sets. If the database is a multibyte universal character set, such as UTF-8, the NLS_LANG parameter must still be set to a compatible single-byte character set when compiling the thesaurus.
- Adding a knowledge base works best for whitespace delimited languages.
- You can have at most one knowledge base for each Globalization Support language.
- Obtaining hierarchical query feedback information such as broader terms, narrower terms and related terms does not work in languages other than English and French. In other languages, the knowledge bases are derived entirely from your thesauri. In such cases, Oracle recommends that you obtain hierarchical information directly from your thesauri.

Order of Precedence for Multiple Thesauri

When multiple thesauri are to be compiled, precedence is determined by the order in which thesauri are listed in the arguments to the compiler (most preferred first). A user thesaurus always has precedence over the built-in knowledge base.

Size Limits for Extended Knowledge Base

The following table lists the size limits associated with creating and compiling an extended knowledge base:
Lexical Compiler (ctxlc)

The Lexical Compiler (ctxlc) is a command-line utility that enables you to create your own Chinese and Japanese lexicons (dictionaries). Such a lexicon may either be generated from a user-supplied word list or from the merging of a word list with the system lexicon for that language.

ctxlc creates the new lexicon in your current directory. The new lexicon consists of three files, drold.dat, drolk.dat, and droli.dat. To change your system lexicon for Japanese or Chinese, overwrite the system lexicon with these files.

The Lexical Compiler can also generate wordlists from the system lexicons for Japanese and Chinese, enabling you to see their contents. These word lists go to the standard output and thus can be redirected into a file of your choice.

After overwriting the system lexicon, you need to re-create your indexes before querying them.

Syntax of ctxlc

ctxlc has the following syntax:

- `ctxlc -ja | -zh [-n] -ics character_set -i input_file`
- `ctxlc -ja | -zh -ocs character_set [ > output_file ]`

Mandatory Arguments

- `-ja | -zh`

Specify the language of the lexicon to modify or create. `-ja` indicates the Japanese lexicon; `-zh` indicates the Chinese lexicon.

<table>
<thead>
<tr>
<th>Description of Parameter</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of RTs (from + to) for each term</td>
<td>32</td>
</tr>
<tr>
<td>Number of terms for each single hierarchy (for example, all narrower terms for a given top term)</td>
<td>64000</td>
</tr>
<tr>
<td>Number of new terms in an extended knowledge base</td>
<td>1 million</td>
</tr>
<tr>
<td>Number of separate thesauri that can be compiled into a user extension to the KB</td>
<td>16</td>
</tr>
</tbody>
</table>

**Lexical Compiler (ctxlc)**
**-ics character_set**
Specify the character set of the input file denoted by `input_file`. `input_file` is the list of words, one word to a line, to use in creating the new lexicon.

**-i input_file**
Specify the file containing words to use in creating a new lexicon.

**-ocs character_set**
Specify the character set of the text file to be output.

**Optional Arguments**

**-n**
Specify `-n` to create a new lexicon that consists only of user-supplied words taken from `input_file`. If `-n` is not specified, then the new lexicon consists of a merge of the system lexicon with `input_file`. Also, when `-n` is not selected, a text file called `drolt.dat`, is created in the current directory to enable you to inspect the contents of the merged lexicon without having to issue another `ctxlc` command.

**Performance Considerations**

You can add up to 1,000,000 new words to a lexicon. However, creating a very large lexicon can cause a performance hit in indexing and querying. Performance is best when the lexicon character set is UTF-8. There is no performance impact on the Chinese or Japanese V-gram lexers, as they do not use lexicons.

**ctxlc Usage Notes**

Oracle recommends the following practices with regard to `ctxlc`:

- Save your plain text dictionary file in your environment for emergency use.
- When upgrading or downgrading your database to a different release, recompile your plain text dictionary file in the new environment so that the user lexicon will work correctly.

**Example**

In this example, you create a new Japanese lexicon from the file `jadict.txt`, a word list that uses the JA16EUC character set. Because you are not specifying `-n`, the new lexicon is the result of merging `jadict.txt` with the system Japanese lexicon. You then replace the existing Japanese lexicon with the new, merged one.
% ctxlc -ja -ics JA16EUC -i jadict.txt

This creates new files in the current directory:

% ls
drold.dat
drolk.dat
droli.dat
drolt.dat

The system lexicon files for Japanese and Chinese are named droldxx.dat
drolkxx.dat, and drolixx.dat, where xx is either JA (for Japanese) or ZH (for
Chinese). Rename the three new files and copy them to the directory containing the
system Japanese lexicon.

% mv drold.dat droldJA.dat
% mv drolk.dat drolkJA.dat
% mv droli.dat droliJA.dat
% cp *dat $ORACLE_HOME/ctx/data/jalx

This replaces the system Japanese lexicon with one that is a merge of the old system
lexicon and your wordlist from jadict.txt.

You can also use ctxlc to get a dump of a system lexicon. This example dumps
the Chinese lexicon to a file called new_chinese_dict.txt in the current
directory:

% ctxlc -zh -ocs UTF8 > new_chinese_dict.txt

This creates a file, new_japanese.dict.txt, using the UTF8 character set, in the
current directory.
This chapter describes various ways that Oracle Text handles alternative spelling of words. It also documents the alternative spelling conventions that Oracle Text uses in the German, Danish, and Swedish languages.

The following topics are covered:

- Overview of Alternative Spelling Features
- Overriding Alternative Spelling Features
- Alternative Spelling Conventions

Overview of Alternative Spelling Features

Some languages have alternative spelling forms for certain words. For example, the German word Schoen can also be spelled as Schön.

The form of a word is either original or normalized. The original form of the word is how it appears in the source document. The normalized form is how it is transformed, if it is transformed at all. Depending on the word being indexed and which system preferences are in effect (these are discussed in this chapter), the normalized form of a word may be the same as the original form. Also, the normalized form may comprise more than one spelling. For example, the normalized form of Schoen is both Schoen and Schön.

Oracle Text handles indexing of alternative word forms in the following ways:

- Alternate Spelling—indexing of alternative forms is enabled
- Base-Letter Conversion—accented letters are transformed into non-accented representations
- New German Spelling—reformed German spelling is accepted
Overview of Alternative Spelling Features

You enable these features by specifying the appropriate attribute to the BASIC_LEXER. For instance, you enable Alternate Spelling by specifying either GERMAN, DANISH, or SWEDISH for the ALTERNATE_SPELLING attribute. As an example, here is how to enable Alternate Spelling in German:

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

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

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

Oracle Text converts query terms to their normalized forms before lookup. As a result, users can query words with either spelling. If Schoen has been indexed as both Schoen and Schön, a query with Schön returns documents containing either form.

Alternate Spelling

When Swedish, German, or Danish has more than one way of spelling a word, Oracle Text normally indexes the word in its original form; that is, as it appears in the source document.

When Alternate Spelling is enabled, Oracle Text indexes words in their normalized form. So, for example, Schoen is indexed both as Schoen and as Schön, and a query on Schoen will return documents containing either spelling. (The same is true of a query on Schön.)

To enable Alternate Spelling, set the BASIC_LEXER attribute ALTERNATE_SPELLING to GERMAN, DANISH, or SWEDISH. See BASIC_LEXER on page 2-36 for more information.

Base-Letter Conversion

Besides alternative spelling, Oracle Text also handles base-letter conversions. With base-letter conversions enabled, letters with umlauts, acute accents, cedillas, and the like are converted to their basic forms for indexing, so fiancé is indexed both as fiancé and as fiance, and a query of fiancé returns documents containing either form.
To enable base-letter conversions, set the BASIC_LEXER attribute BASE_LETTER to YES. See BASIC_LEXER on page 2-36 for more information.

When Alternate Spelling is also enabled, Base-Letter Conversion may need to be overridden to prevent unexpected results. See Overriding Base-Letter Transformations with Alternate Spelling on page 15-4 for more information.

**Generic Versus Language-Specific Base-Letter Conversions**

The BASE_LETTER_TYPE attribute affects the way base-letter conversions take place. It has two possible values: GENERIC or SPECIFIC.

The GENERIC value is the default and specifies that base letter transformation uses one transformation table that applies to all languages.

The SPECIFIC value means that a base-letter transformation that has been specifically defined for your language will be used. This enables you to use accent-sensitive searches for words in your own language, while ignoring accents that are from other languages.

For example, both the GENERIC and the Spanish SPECIFIC tables will transform é into e. However, they treat the letter ñ distinctly. The GENERIC table treats ñ as an n with an accent (actually, a tilde), and so transforms ñ to n. The Spanish SPECIFIC table treats ñ as a separate letter of the alphabet, and thus does not transform it.

**New German Spelling**

In 1996, new spelling rules for German were approved by representatives from all German-speaking countries. For example, under the spelling reforms, Potential becomes Potenzial, Schiffahrt becomes Schifffahrt, and schneuzen becomes schnäuzen.

When the BASIC_LEXER attribute NEW_GERMAN_SPELLING is set to YES, then a CONTAINS query on a German word that has both new and traditional forms will return documents matching both forms. For example, a query on Potential returns documents containing both Potential and Potenzial. The default setting is NO.

---

**Note:** Under reformed German spelling, many words traditionally spelled as one word, such as soviel, are now spelled as two (so viel). Currently, Oracle Text does not make these conversions, nor conversions from two words to one (for example, weh tun to wehtun).
The case of the transformed word is determined from the first two characters of the word in the source document; that is, `schiffahrt` becomes `schiffahrt`, `Schiffahrt` becomes `Schiffahrt`, and `SCHIFFAHRT` becomes `SCHIFFFAHRT`.

As many new German spellings include hyphens, it is recommended that users choosing `NEW_GERMAN_SPELLING` define hyphens as `printjoin`s.

See `BASIC_LEXER` on page 2-36 for more information on setting this attribute.

**Overriding Alternative Spelling Features**

Even when alternative spelling features have been specified by lexer preference, it is possible to override them. Overriding takes the following form:

- Overriding of base-letter conversion when Alternate Spelling is used, to prevent characters with alternate spelling forms, such as `ü`, `ö`, and `ä`, from also being transformed to the base letter forms.

**Overriding Base-Letter Transformations with Alternate Spelling**

Transformations caused by turning on `alternate_spelling` are performed before those of `base_letter`, which can sometimes cause unexpected results when both are enabled.

When Alternate Spelling is enabled, Oracle Text converts two-letter forms to single-letter forms (for example, `ue` to `ü`), so that words can be searched in both their base and alternate forms. Therefore, with Alternate Spelling enabled, a search for `Schoen` will return documents with both `Schoen` and `Schön`.

However, when Base-letter Transformation is also enabled, the `ö` in `Schön` is transformed into an `o`, producing the non-existent word (in German, anyway) `Schon`, and the word is indexed in all three forms.

To prevent this secondary conversion, set the `OVERRIDE_BASE_LETTER` attribute to `TRUE`.

`OVERRIDE_BASE_LETTER` only affects letters with umlauts; accented letters, for example, are still transformed into their base forms.

For more on `BASE_LETTER`, see `Base-Letter Conversion` on page 15-2.

**Alternative Spelling Conventions**

The following sections show the alternative spelling substitutions used by Oracle Text.
German Alternate Spelling Conventions

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

<table>
<thead>
<tr>
<th>Character</th>
<th>Alternate Spelling Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>ä</td>
<td>ae</td>
</tr>
<tr>
<td>ü</td>
<td>ue</td>
</tr>
<tr>
<td>ö</td>
<td>oe</td>
</tr>
<tr>
<td>Ä</td>
<td>AE</td>
</tr>
<tr>
<td>Ü</td>
<td>UE</td>
</tr>
<tr>
<td>Ö</td>
<td>OE</td>
</tr>
<tr>
<td>ß</td>
<td>ss</td>
</tr>
</tbody>
</table>

Danish Alternate Spelling Conventions

The Danish alphabet is the Latin alphabet without the w, plus the special characters: ø æ å. The following table lists the alternate spelling conventions Oracle Text uses for these characters.

<table>
<thead>
<tr>
<th>Character</th>
<th>Alternate Spelling Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>æ</td>
<td>ae</td>
</tr>
<tr>
<td>ø</td>
<td>oe</td>
</tr>
<tr>
<td>å</td>
<td>aa</td>
</tr>
<tr>
<td>Æ</td>
<td>AE</td>
</tr>
<tr>
<td>Ø</td>
<td>OE</td>
</tr>
<tr>
<td>Å</td>
<td>AA</td>
</tr>
</tbody>
</table>

Swedish Alternate Spelling Conventions

The Swedish alphabet is the English alphabet without the w, plus the additional characters: å ä ö. The following table lists the alternate spelling conventions Oracle Text uses for these characters.
<table>
<thead>
<tr>
<th>Old Spelling</th>
<th>New (Reformed) Spelling</th>
</tr>
</thead>
<tbody>
<tr>
<td>å</td>
<td>ae</td>
</tr>
<tr>
<td>å</td>
<td>aa</td>
</tr>
<tr>
<td>ö</td>
<td>oe</td>
</tr>
<tr>
<td>Ä</td>
<td>AE</td>
</tr>
<tr>
<td>Ä</td>
<td>AA</td>
</tr>
<tr>
<td>Ö</td>
<td>OE</td>
</tr>
</tbody>
</table>
This appendix describes the structure of the result tables used to store the output generated by the procedures in the CTX_QUERY, CTX_DOC, and CTX_THES packages.

The following topics are discussed in this appendix:

- CTX_QUERY Result Tables
- CTX_DOC Result Tables
- CTX_THES Result Tables and Data Types

**CTX_QUERY Result Tables**

For the CTX_QUERY procedures that return results, tables for storing the results must be created before the procedure is called. The tables can be named anything, but must include columns with specific names and data types.

This section describes the following types of result tables, and their required columns:

- EXPLAIN Table
- HFEEDBACK Table

**EXPLAIN Table**

Table A–1 describes the structure of the table to which CTX_QUERY.EXPLAIN writes its results.
Table A–1  EXPLAIN Result Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLAIN_ID</td>
<td>VARCHAR2 (30)</td>
<td>The value of the explain_id argument specified in the FEEDBACK call.</td>
</tr>
<tr>
<td>ID</td>
<td>NUMBER</td>
<td>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.</td>
</tr>
<tr>
<td>PARENT_ID</td>
<td>NUMBER</td>
<td>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.</td>
</tr>
<tr>
<td>OPERATION</td>
<td>VARCHAR2 (30)</td>
<td>Name of the internal operation performed. Refer to Table A–2 for possible values.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>VARCHAR2 (30)</td>
<td>Characters that describe a variation on the operation described in the OPERATION column. When an OPERATION has more than one OPTIONS associated with it, OPTIONS values are concatenated in the order of processing. See Table A–3 for possible values.</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>VARCHAR2 (80)</td>
<td>Section name, wildcard term, weight, or threshold value or term to lookup in the index.</td>
</tr>
<tr>
<td>POSITION</td>
<td>NUMBER</td>
<td>The order of processing for nodes that all have the same PARENT_ID. The positions are numbered in ascending order starting at 1.</td>
</tr>
<tr>
<td>CARDINALITY</td>
<td>NUMBER</td>
<td>Reserved for future use. You should create this column for forward compatibility.</td>
</tr>
</tbody>
</table>

Operation Column Values

Table A–2 shows the possible values for the OPERATION column of the EXPLAIN table.

Table A–2  EXPLAIN Table OPERATION Column

<table>
<thead>
<tr>
<th>Operation Value</th>
<th>Query Operator</th>
<th>Equivalent Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABOUT</td>
<td>ABOUT</td>
<td>(none)</td>
</tr>
<tr>
<td>ACCUMULATE</td>
<td>ACCUM</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>AND</td>
<td>&amp;</td>
</tr>
</tbody>
</table>
The following table lists the possible values for the OPTIONS column of the EXPLAIN table.

### Table A–3 EXPLAIN Table OPTIONS Column

<table>
<thead>
<tr>
<th>Options Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>($)</td>
<td>Stem</td>
</tr>
<tr>
<td>(?)</td>
<td>Fuzzy</td>
</tr>
<tr>
<td>(!)</td>
<td>Soundex</td>
</tr>
<tr>
<td>(T)</td>
<td>Order for ordered Near.</td>
</tr>
<tr>
<td>(F)</td>
<td>Order for unordered Near.</td>
</tr>
<tr>
<td>(n)</td>
<td>A number associated with the max_span parameter for the Near operator.</td>
</tr>
</tbody>
</table>
HFEEDBACK Table

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

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEEDBACK_ID</td>
<td>VARCHAR2(30)</td>
<td>The value of the feedback_id argument specified in the HFEEDBACK call.</td>
</tr>
<tr>
<td>ID</td>
<td>NUMBER</td>
<td>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.</td>
</tr>
<tr>
<td>PARENT_ID</td>
<td>NUMBER</td>
<td>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.</td>
</tr>
<tr>
<td>OPERATION</td>
<td>VARCHAR2(30)</td>
<td>Name of the internal operation performed. Refer to Table A–5 for possible values.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>VARCHAR2(30)</td>
<td>Characters that describe a variation on the operation described in the OPERATION column. When an OPERATION has more than one OPTIONS associated with it, OPTIONS values are concatenated in the order of processing. See Table A–6 for possible values.</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>VARCHAR2(80)</td>
<td>Section name, wildcard term, weight, threshold value or term to lookup in the index.</td>
</tr>
<tr>
<td>POSITION</td>
<td>NUMBER</td>
<td>The order of processing for nodes that all have the same PARENT_ID. The positions are numbered in ascending order starting at 1.</td>
</tr>
<tr>
<td>BT_FEEDBACK</td>
<td>CTX_FEEDBACK_TYPE</td>
<td>Stores broader feedback terms. See Table A–7.</td>
</tr>
<tr>
<td>PT_FEEDBACK</td>
<td>CTX_FEEDBACK_TYPE</td>
<td>Stores related feedback terms. See Table A–7.</td>
</tr>
<tr>
<td>NT_FEEDBACK</td>
<td>CTX_FEEDBACK_TYPE</td>
<td>Stores narrower feedback terms. See Table A–7.</td>
</tr>
</tbody>
</table>
Operation Column Values

Table A–5 shows the possible values for the OPERATION column of the HFEEDBACK table.

<table>
<thead>
<tr>
<th>Operation Value</th>
<th>Query Operator</th>
<th>Equivalent Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABOUT</td>
<td>ABOUT</td>
<td>(none)</td>
</tr>
<tr>
<td>ACCUMULATE</td>
<td>ACCUM</td>
<td>,</td>
</tr>
<tr>
<td>AND</td>
<td>AND</td>
<td>&amp;</td>
</tr>
<tr>
<td>EQUIVALENCE</td>
<td>EQUIV</td>
<td>=</td>
</tr>
<tr>
<td>MINUS</td>
<td>MINUS</td>
<td>–</td>
</tr>
<tr>
<td>NEAR</td>
<td>NEAR</td>
<td>;</td>
</tr>
<tr>
<td>NOT</td>
<td>NOT</td>
<td>~</td>
</tr>
<tr>
<td>OR</td>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>SECTION</td>
<td>(section)</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>word or phrase of a text query</td>
<td></td>
</tr>
<tr>
<td>THEME</td>
<td>word or phrase of an ABOUT query</td>
<td></td>
</tr>
<tr>
<td>THRESHOLD</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>WITHIN</td>
<td>within</td>
<td>(none)</td>
</tr>
</tbody>
</table>

OPTIONS Column Values

The following table lists the values for the OPTIONS column of the HFEEDBACK table.

<table>
<thead>
<tr>
<th>Options Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T)</td>
<td>Order for ordered Near.</td>
</tr>
<tr>
<td>(F)</td>
<td>Order for unordered Near.</td>
</tr>
<tr>
<td>(n)</td>
<td>A number associated with the max_span parameter for the Near operator.</td>
</tr>
</tbody>
</table>
**CTX_FEEDBACK_TYPE**

The **CTX_FEEDBACK_TYPE** is a nested table of objects. This datatype is pre-defined in the CTXSYS schema. Use this type to define the columns **BT_FEEDBACK**, **RT_FEEDBACK**, and **NT_FEEDBACK**.

The nested table **CTX_FEEDBACK_TYPE** holds objects of type **CTX_FEEDBACK_ITEM_TYPE**, which is also pre-defined in the CTXSYS schema. This object is defined with three members and one method as follows:

*Table A–7  **CTX_FEEDBACK_ITEM_TYPE***

<table>
<thead>
<tr>
<th>Members and Methods</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>member</td>
<td>Feedback term.</td>
</tr>
<tr>
<td>cardinality</td>
<td>member</td>
<td>(reserved for future use.)</td>
</tr>
<tr>
<td>score</td>
<td>member</td>
<td>(reserved for future use.)</td>
</tr>
</tbody>
</table>

The SQL code that defines these objects is as follows:

```sql
CREATE OR REPLACE TYPE ctx_feedback_type AS TABLE OF ctx_feedback_item_type;
CREATE OR REPLACE TYPE ctx_feedback_item_type AS OBJECT
  (text        VARCHAR2(80),
   cardinality NUMBER,
   score       NUMBER,
   MAP MEMBER FUNCTION rank RETURN REAL,
   PRAGMA RESTRICT_REFERENCES (rank, RNDS, WNDS, RNPS, WNPS)
  );
CREATE OR REPLACE TYPE BODY ctx_feedback_item_type AS
  MAP MEMBER FUNCTION rank RETURN REAL IS
    BEGIN
      RETURN score;
    END rank;
END;
```

**See Also:** For an example of how to select from the HFEEDBACK table and its nested tables, refer to **CTX_QUERY.HFEEDBACK** in Chapter 10, "CTX_QUERY Package".
CTX_DOC Result Tables

The CTX_DOC procedures return results stored in a table. Before calling a procedure, you must create the table. The tables can be named anything, but must include columns with specific names and data types.

This section describes the following result tables and their required columns:

- Filter Table
- Gist Table
- Highlight Table
- Markup Table
- Theme Table

Filter Table

A filter table stores one row for each filtered document returned by CTX_DOC.FILTER. Filtered documents can be plain text or HTML.

When you call CTX_DOC.FILTER for a document, the document is processed through the filter defined for the text column and the results are stored in the filter table you specify.

Filter tables can be named anything, but must include the following columns, with names and datatypes as specified:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_ID</td>
<td>NUMBER</td>
<td>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)</td>
</tr>
<tr>
<td>DOCUMENT</td>
<td>CLOB</td>
<td>Text of the document, stored in plain text or HTML.</td>
</tr>
</tbody>
</table>

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:
Highlight Table

A highlight table stores offset and length information for highlighted terms in a document. This information is generated by CTX_DOC.HIGHLIGHT. Highlighted terms can be the words or phrases that satisfy a word or an ABOUT query.

If a document is formatted, the text is filtered into either plain text or HTML and the offset information is generated for the filtered text. The offset information can be used to highlight query terms for the same document filtered with CTX_DOC.FILTER.

Highlight tables can be named anything, but must include the following columns, with names and datatypes as specified:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_ID</td>
<td>NUMBER</td>
<td>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)</td>
</tr>
<tr>
<td>OFFSET</td>
<td>NUMBER</td>
<td>The position of the highlight in the document, relative to the start of document which has a position of 1.</td>
</tr>
<tr>
<td>LENGTH</td>
<td>NUMBER</td>
<td>The length of the highlight.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_ID</td>
<td>NUMBER</td>
<td>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)</td>
</tr>
<tr>
<td>OFFSET</td>
<td>NUMBER</td>
<td>The position of the highlight in the document, relative to the start of document which has a position of 1.</td>
</tr>
<tr>
<td>LENGTH</td>
<td>NUMBER</td>
<td>The length of the highlight.</td>
</tr>
</tbody>
</table>

Table A–9  Gist Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_ID</td>
<td>NUMBER</td>
<td>Query ID.</td>
</tr>
<tr>
<td>POV</td>
<td>VARCHAR2(80)</td>
<td>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.</td>
</tr>
<tr>
<td>GIST</td>
<td>CLOB</td>
<td>Text of Gist or theme summary, stored as plain text</td>
</tr>
</tbody>
</table>
Markup tables can be named anything, but must include the following columns, with names and datatypes as specified:

**Table A–11  Markup Table**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_ID</td>
<td>NUMBER</td>
<td>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)</td>
</tr>
<tr>
<td>DOCUMENT</td>
<td>CLOB</td>
<td>Marked-up text of the document, stored in plain text or HTML format</td>
</tr>
</tbody>
</table>

**Theme Table**

A theme table stores one row for each theme generated by CTX_DOC.THEMES. The value stored in the THEME column is either a single theme phrase or a string of parent themes, separated by colons.

Theme tables can be named anything, but must include the following columns, with names and data types as specified:

**Table A–12  Theme Table**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_ID</td>
<td>NUMBER</td>
<td>Query ID</td>
</tr>
<tr>
<td>THEME</td>
<td>VARCHAR2 (2000)</td>
<td>Theme phrase or string of parent themes separated by colons (:).</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>NUMBER</td>
<td>Weight of theme phrase relative to other theme phrases for the document.</td>
</tr>
</tbody>
</table>

**Token Table**

A token table stores the text tokens for a document as output by the CTX_DOC.TOKENS procedure. Token tables can be named anything, but must include the following columns, with names and data types as specified.
The CTX.THES expansion functions such as BT, NT, and SYN can return the expansions in a table of type EXP_TAB. You can specify the name of your table with the restab argument.

**EXP_TAB Table Type**

The EXP_TAB table type is a table of rows of type EXP_REC.

The EXP_REC and EXP_TAB types are defined as follows in the CTXSYS schema:

```plsql
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:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_ID</td>
<td>NUMBER</td>
<td>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)</td>
</tr>
<tr>
<td>TOKEN</td>
<td>VARCHAR2(64)</td>
<td>The token string in the text.</td>
</tr>
<tr>
<td>OFFSET</td>
<td>NUMBER</td>
<td>The position of the token in the document, relative to the start of document which has a position of 1.</td>
</tr>
<tr>
<td>LENGTH</td>
<td>NUMBER</td>
<td>The character length of the token.</td>
</tr>
</tbody>
</table>
**EXP_REC Field** | **Description**
--- | ---
xrel | The xrel field contains the relation of the term to the input term (for example, 'SYN', 'PT', 'RT', and so on). The xrel value is PHRASE when the input term appears in the expansion. For translations, the xrel value is the language.  
| |  
xlevel | The xlevel field is the level of the relation. This is used mainly when xrel is a hierarchical relation (BT*/NT*).  
The xlevel field is 0 when xrel is PHRASE.  
The xlevel field is 2 for translations of synonyms under TRSYN.  
The xlevel field is 1 for operators that are not hierarchical, such as PT and RT.  
| |  
xphrase | The xphrase is the related term. This includes a qualifier in parentheses, if one exists for the related term. Compound terms are not de-compounded.  
| |  

Supported Document Formats

This appendix contains a list of the document formats supported by the Inso filtering technology. The following topics are covered in this appendix:

- About Document Filtering Technology
- Supported Document Formats
- Restrictions on Format Support

About Document Filtering Technology

Oracle Text uses document filtering technology licensed from Stellent Chicago, Inc. This filtering technology enables you to index most document formats. This technology also enables you to convert documents to HTML for document presentation with the CTX_DOC package. The software is based in part on the work of the Independent JPEG Group.

See Also: For a list of supported formats, see "Supported Document Formats" on page B-3.

To use Inso filtering for indexing and DML processing, you must specify the INSO_FILTER object in your filter preference.

To use Inso filtering technology for converting documents to HTML with the CTX_DOC package, you need not use the INSO_FILTER indexing preference, but you must still set up your environment to use this filtering technology as described in this appendix.

To convert documents to HTML format, Inso filtering technology relies on shared libraries and data files licensed from Stellent Chicago, Inc.
The following sections discuss the supported platforms and how to enable Inso filtering on the different platforms.

**Latest Updates for Patch Releases**

The supported platforms and formats listed in this appendix apply for this release. These supported formats are updated for patch releases. To view the latest formats, refer to the Oracle Technology Network:

http://otn.oracle.com/products/text/content.html

**Supported Platforms**

Several platforms can take advantage of Inso filter technology.

**Supported Platforms**

Inso filter technology is supported on the following platforms:

- Sun Solaris on SPARC 32-bit and 64-bit (6 - 9.0)
- IBM AIX 32-bit and 64-bit (4.3, 5.1, 5.2)
- HP-UX 32-bit and 64-bit (10.0 - 11.0)
- Red Hat Linux on Intel x86 (7.1, 7.2, 8.0, 9.0)
- SuSE Linux on Intel x86 (7.x and 8.x)
- Microsoft Windows (32-bit)
  - Windows NT (4.0 and above)
  - Windows 95
  - Windows 98
  - Windows 98SE
  - Windows ME
  - Windows 2000
  - Windows XP
  - Windows 2003
- Microsoft Windows (64-bit)
Environment Variables

All environment variables related to Inso filtering must be made visible to Oracle Text.

Requirements for UNIX Platforms

The following requirements apply to Solaris, IBM AIX, HP/UX, and Linux platforms:

- Set the $HOME environment variable to enable Inso technology to write files to a subdirectory (.oit) in $HOME directory.

Supported Document Formats

The tables in this section list the document formats that Oracle Text supports for filtering. Document filtering is used for indexing, DML, and for converting documents to HTML with the CTX_DOC package. This filtering technology is based on Outside In HTML Export and Outside In Viewer Technology, licensed from Stellent Chicago, Inc.

---

**Note:** These lists do not represent the complete list of formats that Oracle Text is able to process. The external filter framework enables Oracle Text to process *any* document format, provided an external filter exists that can filter all the formats to text.

---

**Word Processing Formats - Generic Text**

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII Text</td>
<td>7- &amp; 8-bit</td>
</tr>
<tr>
<td>ANSI Text</td>
<td>7- &amp; 8-bit</td>
</tr>
<tr>
<td>Unicode Text</td>
<td>All versions</td>
</tr>
<tr>
<td>HTML</td>
<td>Versions through 3.0 (some limitations)</td>
</tr>
<tr>
<td>IBM Revisable Form Text</td>
<td>All versions</td>
</tr>
<tr>
<td>IBM FFT</td>
<td>All versions</td>
</tr>
<tr>
<td>Microsoft Rich Text Format (RTF)</td>
<td>All versions</td>
</tr>
</tbody>
</table>
## Supported Document Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>WML</td>
<td>Version 5.2</td>
</tr>
</tbody>
</table>

### Word Processing Formats - DOS

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC WPS Plus (WPL)</td>
<td>Versions through 4.1</td>
</tr>
<tr>
<td>DEC WPS Plus (DX)</td>
<td>Versions through 4.0</td>
</tr>
<tr>
<td>DisplayWrite 2 &amp; 3 (TXT)</td>
<td>All versions</td>
</tr>
<tr>
<td>DisplayWrite 4 &amp; 5</td>
<td>Versions through Release 2.0</td>
</tr>
<tr>
<td>Enable</td>
<td>Versions 3.0, 4.0 and 4.5</td>
</tr>
<tr>
<td>First Choice</td>
<td>Versions through 3.0</td>
</tr>
<tr>
<td>Framework</td>
<td>Version 3.0</td>
</tr>
<tr>
<td>IBM Writing Assistant</td>
<td>Version 1.01</td>
</tr>
<tr>
<td>Lotus Manuscript</td>
<td>Version 2.0</td>
</tr>
<tr>
<td>MASS11</td>
<td>Versions through 8.0</td>
</tr>
<tr>
<td>Microsoft Word</td>
<td>Versions through 6.0</td>
</tr>
<tr>
<td>Microsoft Works</td>
<td>Versions through 2.0</td>
</tr>
<tr>
<td>MultiMate</td>
<td>Versions through 4.0</td>
</tr>
<tr>
<td>Navy DIF</td>
<td>All versions</td>
</tr>
<tr>
<td>Nota Bene</td>
<td>Version 3.0</td>
</tr>
<tr>
<td>Novell Word Perfect</td>
<td>Versions through 6.1</td>
</tr>
<tr>
<td>Office Writer</td>
<td>Version 4.0 to 6.0</td>
</tr>
<tr>
<td>PC-File Letter</td>
<td>Versions through 5.0</td>
</tr>
<tr>
<td>PC-File+ Letter</td>
<td>Versions through 3.0</td>
</tr>
<tr>
<td>PFS:Write</td>
<td>Versions A, B, and C</td>
</tr>
<tr>
<td>Professional Write</td>
<td>Versions through 2.1</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>Version 2.0</td>
</tr>
<tr>
<td>Samna Word</td>
<td>Versions through Samna Word IV+</td>
</tr>
</tbody>
</table>
### Word Processing Formats - Windows

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hangul</td>
<td>Version 97</td>
</tr>
<tr>
<td>Novell/Corel WordPerfect for Windows</td>
<td>Versions through 10</td>
</tr>
<tr>
<td>JustWrite</td>
<td>Versions through 3.0</td>
</tr>
<tr>
<td>JustSystems Ichitaro</td>
<td>Version 5.0, 6.0, 8.0, 9.0, and 10.0</td>
</tr>
<tr>
<td>Legacy</td>
<td>Versions through 1.1</td>
</tr>
<tr>
<td>Lotus AMI/AMI Professional</td>
<td>Versions through 3.1</td>
</tr>
<tr>
<td>Lotus WordPro (Non-32-bit-Windows platforms are Text-only)</td>
<td>Version 96 through Millennium Edition 9.6</td>
</tr>
<tr>
<td>Microsoft Works for Windows</td>
<td>Versions through 4.0</td>
</tr>
<tr>
<td>Microsoft Windows Write</td>
<td>Versions through 3.0</td>
</tr>
<tr>
<td>Microsoft Word for Windows</td>
<td>Versions through 2002</td>
</tr>
<tr>
<td>Microsoft WordPad</td>
<td>All versions</td>
</tr>
<tr>
<td>Novell Perfect Works</td>
<td>Version 2.0</td>
</tr>
<tr>
<td>Professional Write Plus</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>Q&amp;A Write for Windows</td>
<td>Version 3.0</td>
</tr>
</tbody>
</table>
## Supported Document Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>StarOffice Writer for Windows and UNIX (Text only)</td>
<td>Version 5.2</td>
</tr>
<tr>
<td>WordStar for Windows</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>Adobe FrameMaker (MIF)</td>
<td>Version 6.0</td>
</tr>
</tbody>
</table>

### Word Processing Formats - Macintosh

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Word for Mac</td>
<td>Versions 3.0 - 4.0, 98, 2001</td>
</tr>
<tr>
<td>Novell WordPerfect</td>
<td>Versions 1.02 through 3.0</td>
</tr>
<tr>
<td>Microsoft Works for Mac</td>
<td>Versions through 2.0</td>
</tr>
<tr>
<td>MacWrite II</td>
<td>Version 1.1</td>
</tr>
</tbody>
</table>

### Spreadsheet Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Versions 3.0, 4.0 and 4.5</td>
</tr>
<tr>
<td>First Choice</td>
<td>Versions through 3.0</td>
</tr>
<tr>
<td>Framework</td>
<td>Version 3.0</td>
</tr>
<tr>
<td>Lotus 1-2-3 (DOS &amp; Windows)</td>
<td>Versions through 5.0</td>
</tr>
<tr>
<td>Lotus 1-2-3 for SmartSuite</td>
<td>Version 97 - Millennium 9.6</td>
</tr>
<tr>
<td>Lotus 1-2-3 Charts (DOS &amp; Windows)</td>
<td>Versions through 5.0</td>
</tr>
<tr>
<td>Lotus 1-2-3 (OS/2)</td>
<td>Versions through 2.0</td>
</tr>
<tr>
<td>Lotus Symphony</td>
<td>Versions 1.0, 1.1 and 2.0</td>
</tr>
<tr>
<td>Microsoft Excel Windows</td>
<td>Versions 2.2 through 2002</td>
</tr>
<tr>
<td>Microsoft Excel Macintosh</td>
<td>Versions 3.0 - 4.0, 98 and 2001</td>
</tr>
<tr>
<td>Microsoft Excel Charts</td>
<td>Versions 2.x - 7.0</td>
</tr>
<tr>
<td>Microsoft Multiplan</td>
<td>Version 4.0</td>
</tr>
<tr>
<td>Microsoft Works for Windows</td>
<td>Versions through 4.0</td>
</tr>
</tbody>
</table>
### Supported Document Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Works (DOS)</td>
<td>Versions through 2.0</td>
</tr>
<tr>
<td>Microsoft Works (Mac)</td>
<td>Versions through 2.0</td>
</tr>
<tr>
<td>Mosaic Twin</td>
<td>Version 2.5</td>
</tr>
<tr>
<td>Novell Perfect Works</td>
<td>Version 2.0</td>
</tr>
<tr>
<td>Quattro Pro for DOS</td>
<td>Versions through 5.0</td>
</tr>
<tr>
<td>Quattro Pro for Windows</td>
<td>Versions through 10</td>
</tr>
<tr>
<td>PFS:Professional Plan</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>SuperCalc 5</td>
<td>Version 4.0</td>
</tr>
<tr>
<td>SmartWare II</td>
<td>Version 1.02</td>
</tr>
<tr>
<td>StarOffice Calc for Windows and UNIX</td>
<td>Version 5.2</td>
</tr>
<tr>
<td>VP Planner 3D</td>
<td>Version 1.0</td>
</tr>
</tbody>
</table>

### Database Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Versions through 2.0</td>
</tr>
<tr>
<td>dBASE</td>
<td>Versions through 5.0</td>
</tr>
<tr>
<td>DataEase</td>
<td>Version 4.x</td>
</tr>
<tr>
<td>dBXL</td>
<td>Version 1.3</td>
</tr>
<tr>
<td>Enable</td>
<td>Versions 3.0, 4.0 and 4.5</td>
</tr>
<tr>
<td>First Choice</td>
<td>Versions through 3.0</td>
</tr>
<tr>
<td>FoxBase</td>
<td>Version 2.1</td>
</tr>
<tr>
<td>Framework</td>
<td>Version 3.0</td>
</tr>
<tr>
<td>Microsoft Works for Windows</td>
<td>Versions through 4.0</td>
</tr>
<tr>
<td>Microsoft Works (DOS)</td>
<td>Versions through 2.0</td>
</tr>
<tr>
<td>Microsoft Works (Mac)</td>
<td>Versions through 2.0</td>
</tr>
<tr>
<td>Paradox (DOS)</td>
<td>Versions through 4.0</td>
</tr>
</tbody>
</table>
### Display Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDF - Portable Document Format</td>
<td>Adobe Acrobat Versions through 5.0 including Chinese (simplified and traditional), Japanese, Korean, and read-only PDF</td>
</tr>
<tr>
<td></td>
<td>Encrypted (password protected) PDF is not supported.</td>
</tr>
<tr>
<td></td>
<td>PDF containing embedded fonts without included character mapping is partially supported: characters that are represented by means of embedded fonts without included character mapping show up as meaningless output; however, all remaining characters (if any) in such a PDF document are still filtered correctly.</td>
</tr>
</tbody>
</table>

### Presentation Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corel/Novell Presentations</td>
<td>Versions through 10</td>
</tr>
<tr>
<td>Harvard Graphics for DOS</td>
<td>Versions 2.x &amp; 3.x</td>
</tr>
<tr>
<td>Harvard Graphics for Windows</td>
<td>Windows versions</td>
</tr>
<tr>
<td>Freelance for Windows</td>
<td>Versions through Millennium 9.6</td>
</tr>
<tr>
<td>Freelance for OS/2</td>
<td>Versions through 2.0</td>
</tr>
</tbody>
</table>
Graphic Formats

The following table lists the graphic formats that the INSO filter recognizes. This means that indexing a text column that contains any of these formats produces no error. As such, it is safe for the column to contain any of these formats.

Note: The INSO filter cannot extract textual information from graphics.

Table B–1 Supported Graphics Formats for INSO Filter

<table>
<thead>
<tr>
<th>Graphics Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adobe Photoshop (PSD)</td>
<td>Version 4.0</td>
</tr>
<tr>
<td>Adobe Illustrator</td>
<td>Versions through 7.0, 9.0</td>
</tr>
<tr>
<td>Adobe FrameMaker graphics (FMV)</td>
<td>Vector/raster through 5.0</td>
</tr>
<tr>
<td>Ami Draw (SDW)</td>
<td>Ami Draw</td>
</tr>
<tr>
<td>AutoCAD Interchange and Native Drawing formats (DXF and DWG)</td>
<td>AutoCAD Drawing Versions 2.5-2.6, 9.0 - 14.0, 2000i and 2002</td>
</tr>
<tr>
<td>AutoShade Rendering (RND)</td>
<td>Version 2.0</td>
</tr>
<tr>
<td>Binary Group 3 Fax</td>
<td>All versions</td>
</tr>
<tr>
<td>Bitmap (BMP, RLE, ICO, CUR, OS/2 DIB &amp; WARP)</td>
<td>No specific version</td>
</tr>
<tr>
<td>CALS Raster (GP4)</td>
<td>Type I and Type II</td>
</tr>
<tr>
<td>Corel Clipart format (CMX)</td>
<td>Versions 5 through 6</td>
</tr>
<tr>
<td>Corel Draw (CDR)</td>
<td>Versions 6.0 - 8.0</td>
</tr>
<tr>
<td>Corel Draw (CDR with TIFF header)</td>
<td>Versions 2.0 - 9.0</td>
</tr>
<tr>
<td>Computer Graphics Metafile (CGM)</td>
<td>ANSI, CALS NIST version 3.0</td>
</tr>
<tr>
<td>Encapsulated PostScript (EPS)</td>
<td>TIFF header only</td>
</tr>
</tbody>
</table>
### Table B–1 Supported Graphics Formats for INSO Filter

<table>
<thead>
<tr>
<th>Graphics Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics Environment Manager (GEM)</td>
<td>Bitmap &amp; vector</td>
</tr>
<tr>
<td>GEM Paint (IMG)</td>
<td>No specific version</td>
</tr>
<tr>
<td>Graphics Interchange Format (GIF)</td>
<td>No specific version</td>
</tr>
<tr>
<td>Hewlett Packard Graphics Language (HPGL)</td>
<td>Version 2</td>
</tr>
<tr>
<td>IBM Graphics Data Format (GDF)</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>IBM Picture Interchange Format (PIF)</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>Initial Graphics Exchange Spec (IGES)</td>
<td>Version 5.1</td>
</tr>
<tr>
<td>JFIF (JPEG not in TIFF format)</td>
<td>All versions</td>
</tr>
<tr>
<td>JPEG (Including EXIF)</td>
<td>No specific version</td>
</tr>
<tr>
<td>Kodak Flash Pix (FPX)</td>
<td>No specific version</td>
</tr>
<tr>
<td>Kodak Photo CD (PCD)</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>Lotus Snapshot</td>
<td>All versions</td>
</tr>
<tr>
<td>Lotus PIC</td>
<td>No specific version</td>
</tr>
<tr>
<td>Macintosh PICT1 &amp; PICT2</td>
<td>Bitmap only</td>
</tr>
<tr>
<td>MacPaint (PNTG)</td>
<td>No specific version</td>
</tr>
<tr>
<td>Micrografx Draw (DRW)</td>
<td>Versions through 4.0</td>
</tr>
<tr>
<td>Micrografx Designer (DRW)</td>
<td>Versions through 3.1</td>
</tr>
<tr>
<td>Micrografx Designer (DSF)</td>
<td>Windows 95, version 6.0</td>
</tr>
<tr>
<td>Novell PerfectWorks (Draw)</td>
<td>Version 2.0</td>
</tr>
<tr>
<td>OS/2 PM Metafile (MET)</td>
<td>Version 3.0</td>
</tr>
<tr>
<td>Paint Shop Pro 6 (PSP) (Windows platform only)</td>
<td>Versions 5.0 - 6.0</td>
</tr>
<tr>
<td>PC Paintbrush (PCX and DCX)</td>
<td>No specific version</td>
</tr>
<tr>
<td>Portable Bitmap (PBM)</td>
<td>All versions</td>
</tr>
<tr>
<td>Portable Graymap (PGM)</td>
<td>No specific version</td>
</tr>
<tr>
<td>Portable Network Graphics (PNG)</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>Portable Pixmap (PPM)</td>
<td>No specific version</td>
</tr>
<tr>
<td>Postscript (PS)</td>
<td>Level II</td>
</tr>
</tbody>
</table>
Table B–1  Supported Graphics Formats for INSO Filter

<table>
<thead>
<tr>
<th>Graphics Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progressive JPEG</td>
<td>No specific version</td>
</tr>
<tr>
<td>Sun Raster (SRS)</td>
<td>No specific version</td>
</tr>
<tr>
<td>TIFF</td>
<td>Versions through 6</td>
</tr>
<tr>
<td>TIFF CCITT Group 3 &amp; 4</td>
<td>Versions through 6</td>
</tr>
<tr>
<td>Truevision TGA (TARGA)</td>
<td>Version 2</td>
</tr>
<tr>
<td>Visio (Preview)</td>
<td>Version 4</td>
</tr>
<tr>
<td>Visio</td>
<td>Versions 5, 2000 and 2002</td>
</tr>
<tr>
<td>WBMP</td>
<td>No specific version</td>
</tr>
<tr>
<td>Windows Enhanced Metafile (EMF)</td>
<td>No specific version</td>
</tr>
<tr>
<td>Windows Metafile (WMF)</td>
<td>No specific version</td>
</tr>
<tr>
<td>WordPerfect Graphics (WPG &amp; WPG2)</td>
<td>Versions through 2.0</td>
</tr>
<tr>
<td>X-Windows Bitmap (XBM)</td>
<td>x10 compatible</td>
</tr>
<tr>
<td>X-Windows Dump (XWD)</td>
<td>x10 compatible</td>
</tr>
<tr>
<td>X-Windows Pixmap (XPM)</td>
<td>x10 compatible</td>
</tr>
</tbody>
</table>

Other Document Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable (EXE, DLL)</td>
<td>No specific version</td>
</tr>
<tr>
<td>Executable for Windows NT</td>
<td>No specific version</td>
</tr>
<tr>
<td>Microsoft Project (Text only)</td>
<td>Version 98</td>
</tr>
<tr>
<td>Microsoft Outlook Message (MSG): (Text only)</td>
<td>No specific version</td>
</tr>
<tr>
<td>vCard</td>
<td>Version 2.1</td>
</tr>
</tbody>
</table>

Restrictions on Format Support

Password-protected documents and documents with password-protected content are not supported by the Inso filter.
This appendix provides examples of how to load text into a text column. It also describes the structure of ctxload import files:

- SQL INSERT Example
- SQL*Loader Example
- Structure of ctxload Thesaurus Import File

SQL INSERT Example

A simple way to populate a text table is to create a table with two columns, id and text, using CREATE TABLE and then use the INSERT statement to load the data. This example makes the id column the primary key, which is optional. The text column is VARCHAR2:

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

To populate the text column, use the INSERT statement as follows:

```sql
insert into docs values(1, 'this is the text of the first document');
insert into docs values(12, 'this is the text of the second document');
```

SQL*Loader Example

The following example shows how to use SQL*Loader to load mixed format documents from the operating system to a BLOB column. The example has two steps:

- create the table
- issue the SQL*Loader command that reads control file and loads data into table
See Also: For a complete discussion on using SQL*Loader, see Oracle9i Database Utilities

Creating the Table

This example loads to a table articles_formatted created as follows:

```sql
CREATE TABLE articles_formatted (
    ARTICLE_ID   NUMBER PRIMARY KEY ,
    AUTHOR       VARCHAR2(30),
    FORMAT       VARCHAR2(30),
    PUB_DATE     DATE,
    TITLE        VARCHAR2(256),
    TEXT         BLOB
);
```

The `article_id` column is the primary key. Documents are loaded in the `text` column, which is of type `BLOB`.

Issuing the SQL*Loader Command

The following command starts the loader, which reads the control file LOADER1.DAT:

```
sqlldr userid=demo/demo control=loader1.dat log=loader.log
```

Example Control File: loader1.dat

This SQL*Loader control file defines the columns to be loaded and instructs the loader to load the data line by line from loader2.dat into the articles_formatted table. Each line in loader2.dat holds a comma separated list of fields to be loaded.

```
-- load file example
load data
INFILE 'loader2.dat'
INTO TABLE articles_formatted
APPEND
FIELDS TERMINATED BY ','
(article_id SEQUENCE (MAX,1),
author CHAR(30),
format,
pub_date SYSDATE,
title,
ext_fname FILLER CHAR(80),
```

C-2 Oracle Text Reference
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 the `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/newslet.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/solvamp.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/oracview.html,
- John Constable, GIF,Laurence J. Ellison : portrait,/sample/docs/larry.gif,
- Alan Greenspan, GIF,Oracle revenues : Graph,/sample/docs/oragraph97.gif,
- Giorgio Armani, GIF,Oracle Revenues : Trend,/sample/docs/oratrend.gif,
Structure of ctxload Thesaurus Import File

The import file must use the following format for entries in the thesaurus:

```
phrase
  BT broader_term
  NT narrower_term1
  NT narrower_term2
  ...
  NT narrower_termN

  BTG broader_term
  NTG narrower_term1
  NTG narrower_term2
  ...
  NTG narrower_termN

  BTP broader_term
  NTP narrower_term1
  NTP narrower_term2
  ...
  NTP narrower_termN

  BTI broader_term
  NTI narrower_term1
  NTI narrower_term2
  ...
  NTI narrower_termN

SYN synonym1
SYN synonym2
...
SYN synonymN

USE synonym1 or SEE synonym1 or PT synonym1
  RT related_term1
  RT related_term2
  ...
  RT related_termN

SN text

language_key: term
```
**Phrase**

is a word or phrase that is defined as having synonyms, broader terms, narrower terms, or related terms.

In compliance with ISO-2788 standards, a TT marker can be placed before a phrase to indicate that the phrase is the top term in a hierarchy; however, the TT marker is not required. In fact, ctxload ignores TT markers during import.

A top term is identified as any phrase that does not have a broader term (BT, BTG, BTP, or BTI).

---

**Note:** The thesaurus query operators (SYN, PT, BT, BTG, BTP, BTI, NT, NTG, NTP, NTI, and RT) are reserved words and, thus, cannot be used as phrases in thesaurus entries.

### BT, BTG, BTP, BTI broader_termN

are the markers that indicate broader_termN is a broader (generic | partitive | instance) term for phrase.

broader_termN is a word or phrase that conceptually provides a more general description or category for phrase. For example, the word *elephant* could have a broader term of *land mammal*.

### NT, NTG, NTP, NTI narrower_termN

are the markers that indicate narrower_termN is a narrower (generic | partitive | instance) term for phrase.

If phrase does not have a broader (generic | partitive | instance) term, but has one or more narrower (generic | partitive | instance) terms, phrase is created as a top term in the respective hierarchy (in an Oracle Text thesaurus, the BT/NT, BTG/NTG, BTP/NTP, and BTI/NTI hierarchies are separate structures).

narrower_termN is a word or phrase that conceptually provides a more specific description for phrase. For example, the word *elephant* could have a narrower terms of *indian elephant* and *african elephant*.

### SYN synonymN

is a marker that indicates phrase and synonymN are synonyms within a synonym ring.

synonymN is a word or phrase that has the same meaning for phrase. For example, the word *dog* could have a synonym of *canine*. 

---

Loading Examples  C-5
Structure of ctxload Thesaurus Import File

Note: Synonym rings are not defined explicitly in Oracle Text thesauri. They are created by the transitive nature of synonyms.

USE SEE PT synonym1
are markers that indicate phrase and synonym1 are synonyms within a synonym ring (similar to SYN).

The markers USE, SEE or PT also indicate synonym1 is the preferred term for the synonym ring. Any of these markers can be used to define the preferred term for a synonym ring.

RT related_termN
is the marker that indicates related_termN is a related term for phrase.

related_termN is a word or phrase that has a meaning related to, but not necessarily synonymous with phrase. For example, the word dog could have a related term of wolf.

Note: Related terms are not transitive. If a phrase has two or more related terms, the terms are related only to the parent phrase and not to each other.

SN text
is the marker that indicates the following text is a scope note (for example, comment) for the preceding entry.

language_key term
term is the translation of phrase into the language specified by language_key.

Alternate Hierarchy Structure
In compliance with thesauri standards, the load file supports formatting hierarchies (BT/NT, BTG/NTG, BTP, NTP, BTI/NTI) by indenting the terms under the top term and using NT (or NTG, NTP, NTI) markers that include the level for the term:

phrase
   NT1 narrower_term1
   NT2 narrower_term1.1
   NT2 narrower_term1.2
   NT3 narrower_term1.2.1
   NT3 narrower_term1.2.2
NT1 narrower_term2
... 
NT1 narrower_termN

Using this method, the entire branch for a top term can be represented hierarchically in the load file.

**Usage Notes for Terms in Import Files**

The following conditions apply to the structure of the entries in the import file:

- each entry (phrase, BT, NT, or SYN) must be on a single line followed by a newline character
- entries can consist of a single word or phrases
- the maximum length of an entry (phrase, BT, NT, or SYN) is 255 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, and so on) must begin with at least one space
- a phrase can occur more than once in the file
- each phrase can have one or more narrower term entries (NT, NTG, NTP), broader term entries (BT, BTG, BTP), synonym entries, and related term entries
- each broader term, narrower term, synonym, and preferred term entry must start with the appropriate marker and the markers must be in capital letters
- the broader terms, narrower terms, and synonyms for a phrase can be in any order
- homographs must be followed by parenthetical disambiguators everywhere they are used
  
  For example: cranes (birds), cranes (lifting equipment)
- compound terms are signified by a plus sign between each factor (for example, buildings + construction)
- compound terms are allowed only as synonyms or preferred terms for other terms, never as terms by themselves, or in hierarchical relations.
- terms can be followed by a scope note (SN), total maximum length of 2000 characters, on subsequent lines
multi-line scope notes are allowed, but require an SN marker on each line of the note

Example of Incorrect SN usage:

VIEW CAMERAS
SN Cameras with through-the lens focusing and a range of movements of the lens plane relative to the film plane

Example of Correct SN usage:

VIEW CAMERAS
SN Cameras with through-the lens focusing and a SN range of movements of the lens plane relative SN to the film plane

Multi-word terms cannot start with reserved words (for example, use is a reserved word, so use other door is not an allowed term; however, use is an allowed term)

Usage Notes for Relationships in Import Files

The following conditions apply to the relationships defined for the entries in the import file:

- related term entries must follow a phrase or another related term entry
- related term entries start with one or more spaces, the RT marker, followed by white space, then the related term on the same line
- multiple related terms require multiple RT markers

Example of incorrect RT usage:

MOVING PICTURE CAMERAS
RT CINE CAMERAS
TELEVISION CAMERAS

Example of correct RT usage:

MOVING PICTURE CAMERAS
RT CINE CAMERAS
RT TELEVISION CAMERAS

Terms are allowed to have multiple broader terms, narrower terms, and related terms
Examples of Import Files

This section provides three examples of correctly formatted thesaurus import files.

Example 1 (Flat Structure)

```plaintext
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)

```plaintext
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
```
Structure of ctxload Thesaurus Import File

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
This Appendix describes the multi-lingual features of Oracle Text. The following topics are discussed:

- Introduction
- Indexing
- Querying
- Supplied Stop Lists
- Knowledge Base
- Multi-Lingual Features Matrix

Introduction

This appendix summarizes the main multilingual features for Oracle Text.

For a complete list of Oracle Globalization Support languages and character set support, refer to the Oracle Database Globalization Support Guide.

Indexing

The following sections describe the multi-lingual indexing features.

Index Types

The following sections describes the supported multilingual features for the Oracle Text index types.
**CONTEXT Index Type**
The CONTEXT index type fully supports multi-lingual features including use of the language and character set columns, use of the MULTI_LEXER, and use of all Chinese, Japanese, and Korean language lexers.

**CTXCAT Index Type**
CTXCAT supports the multi-lingual features of the BASIC_LEXER with the exception of indexing themes.
CTXCAT also supports the following lexers:

- CHINESE_LEXER
- CHINESE_VGRAM_LEXER
- JAPANESE_LEXER
- JAPANESE_VGRAM_LEXER
- KOREAN_LEXER
- KOREAN_MORP_LEXER.

**CTXRULE Index Type**
The CTXRULE index type supports the multi-lingual features of the BASIC_LEXER including ABOUT and STEM operators. It also supports Japanese, Chinese, and Korean.

**Lexer Types**
Oracle Text supports the indexing of different languages by enabling you to choose a lexer in the indexing process. The lexer you employ determines the languages you can index. The following table describes the supported lexers:

<table>
<thead>
<tr>
<th>Lexer</th>
<th>Supported Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASIC_LEXER</td>
<td>English and most western European languages that use white space delimited words.</td>
</tr>
<tr>
<td>MULTI_LEXER</td>
<td>Lexer for indexing tables containing documents of different languages such as English, German, and Japanese.</td>
</tr>
<tr>
<td>CHINESE_VGRAM</td>
<td>Lexer for extracting tokens from Chinese text.</td>
</tr>
</tbody>
</table>
Indexing

Basic Lexer Features

The following features are supported with the BASIC_LEXER preference. You enable these features with attributes of the BASIC_LEXER. Features such as alternate spelling, composite, and base letter can be enabled together for better search results.

Theme Indexing

Enables the indexing and subsequent querying of document concepts with the ABOUT operator with CONTEXT index types. These concepts are derived from the Oracle Text knowledge base. This feature is supported for English and French.

Lexer Supported Languages

<table>
<thead>
<tr>
<th>Lexer</th>
<th>Supported Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHINESE_LEXER</td>
<td>Lexer for extracting tokens from Chinese text. This lexer offers the following benefits over the CHINESE_VGRAM lexer:</td>
</tr>
<tr>
<td></td>
<td>- generates a smaller index</td>
</tr>
<tr>
<td></td>
<td>- better query response time</td>
</tr>
<tr>
<td></td>
<td>- generates real world tokens resulting in better query precision</td>
</tr>
<tr>
<td></td>
<td>- supports stop words</td>
</tr>
<tr>
<td>JAPANESE_VGRAM</td>
<td>Lexer for extracting tokens from Japanese text.</td>
</tr>
<tr>
<td>JAPANESE_LEXER</td>
<td>Lexer for extracting tokens from Japanese text. This lexer offers the following advantages over the JAPANESE_VGRAM lexer:</td>
</tr>
<tr>
<td></td>
<td>- generates smaller index</td>
</tr>
<tr>
<td></td>
<td>- better query response time</td>
</tr>
<tr>
<td></td>
<td>- generates real world tokens resulting in better precision</td>
</tr>
<tr>
<td>KOREAN_LEXER</td>
<td>Lexer for extracting tokens from Korean text.</td>
</tr>
<tr>
<td>KOREAN_MORPH_LEXER</td>
<td>Lexer for extracting tokens from Korean text. This lexer offers the following benefits over the KOREAN_LEXER:</td>
</tr>
<tr>
<td></td>
<td>- better morphological analysis of Korean text</td>
</tr>
<tr>
<td></td>
<td>- faster indexing</td>
</tr>
<tr>
<td></td>
<td>- smaller indexes</td>
</tr>
<tr>
<td></td>
<td>- more accurate query searching</td>
</tr>
<tr>
<td>USER_LEXER</td>
<td>Lexer you create to index a particular language.</td>
</tr>
</tbody>
</table>
This feature is not supported with CTXCAT index types.

**Alternate Spelling**

This feature enables you to search on alternate spellings of words. For example, with alternate spelling enabled in German, a query on `gross` returns documents that contain `groß` and `gross`.

This feature is supported in German, Danish, and Swedish.

Additionally, German can be indexed according to both traditional and reformed spelling conventions.

*See Also:* "Alternate Spelling" on page 15-2 and "New German Spelling" on page 15-3.

**Base Letter Conversion**

This feature enables you to query words with or without diacritical marks such as tildes, accents, and umlauts. For example, with a Spanish base-letter index, a query of `energia` matches documents containing both `energia` and `energía`.

This feature is supported for English and all other supported whitespace delimited languages. In English and French, you can use the basic lexer to enable theme indexing.

*See Also:* "Base-Letter Conversion" on page 15-2

**Composite**

This feature enables you to search on words that contain the specified term as a sub-composite. You must use the stem ($) operator. This feature is supported for German and Dutch.

For example, in German, a query of `$register` finds documents that contain `Bruttoregistertonne` and `Registertonne`.

**Index stems**

This feature enables you to specify a stemmer for stem indexing. Tokens are stemmed to a single base form at index time in addition to the normal forms. Indexing stems enables better query performance for stem queries, such as `$computed`.

This feature is supported for English, Dutch, French, German, Italian, Spanish.
Multi Lexer Features

The MULTI_LEXER lexer enables you to index a column that contains documents of different languages. During indexing Oracle Text examines the language column and switches in the language-specific lexer to process the document. You define the lexer preferences for each language before indexing.

The multi lexer enables you to set different preferences for languages. For example, you can have composite set to TRUE for German documents and composite set to FALSE for Dutch documents.

World Lexer Features

Like MULTI_LEXER, the WORLD_LEXER lexer enables you to index documents that contain different languages; however, it automatically detects the languages of a document and so does not require you to create a language column in the base table.

WORLD_LEXER processes most languages whose characters are defined as part of Unicode 4.0. For WORLD_LEXER to be effective, documents with multiple languages must use AL32UTF-8 or UTF8 Oracle character set encoding (including supplementary, or “surrogate-pair,” characters).

Table D–1 and Table D–2 show the languages supported by WORLD_LEXER. Note: this list may change as the Unicode standard changes, and in any case should not be considered exhaustive. (Languages are group by Unicode writing system, not by natural language groupings.)

Table D–1   Languages Supported by the World Lexer (Space-separated)

<table>
<thead>
<tr>
<th>Language Group</th>
<th>Languages Include</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>Arabic, Farsi, Kurdish, Pashto, Sindhi, Urdu</td>
</tr>
<tr>
<td>Armenian</td>
<td>Armenian</td>
</tr>
<tr>
<td>Bengali</td>
<td>Assamese, Bengali</td>
</tr>
<tr>
<td>Bopomofo</td>
<td>Hakka Chinese, Minnan Chinese</td>
</tr>
<tr>
<td>Cyrillic</td>
<td>Over 50 languages, including Belorussian, Bulgarian,</td>
</tr>
<tr>
<td></td>
<td>Macedonian, Moldavian, Russian, Serbian, Serbo-Croatian,</td>
</tr>
<tr>
<td></td>
<td>Ukrainian</td>
</tr>
<tr>
<td>Devenagari</td>
<td>Bhojpuri, Bihari, Hindi, Kashmiriti, Marathi, Napali,</td>
</tr>
<tr>
<td></td>
<td>Pali, Sanskrit</td>
</tr>
<tr>
<td>Ethiopic</td>
<td>Amharic, Ge’ez, Tigrinya, Tirgre</td>
</tr>
<tr>
<td>Language Group</td>
<td>Languages Include</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Georgian</td>
<td>Georgian</td>
</tr>
<tr>
<td>Greek</td>
<td>Greek</td>
</tr>
<tr>
<td>Gujarati</td>
<td>Gujarati, Kacchi</td>
</tr>
<tr>
<td>Gurmukhi</td>
<td>Punjabi</td>
</tr>
<tr>
<td>Hebrew</td>
<td>Hebrew, Ladino, Yiddish</td>
</tr>
<tr>
<td>Kaganga</td>
<td>Redjang</td>
</tr>
<tr>
<td>Kannada</td>
<td>Kanarese, Kannada</td>
</tr>
<tr>
<td>Korean</td>
<td>Korean, Hanja Hangul</td>
</tr>
<tr>
<td>Latin</td>
<td>Afrikaans, Albanian, Basque, Breton, Catalan, Croatian, Czech, Danish, Dutch, English, Esperanto, Estonian, Faeroese, Fijian, Finnish, Flemish, French, Frisian, German, Hawaiian, Hungarian, Icelandic, Indonesian, Irish, Italian, Lappish, Classic Latin, Latvian, Lithuanian, Malay, Maltese, Pinyin Mandarin, Maori, Norwegian, Polish, Portuguese, Provencal, Romanian, Rumanian, Samoan, Scottish Gaelic, Slovak, Slovene, Slovenian, Sorbian, Spanish, Swahili, Swedish, Tagalog, Turkish, Viennese, Welsh</td>
</tr>
<tr>
<td>Malayalam</td>
<td>Malayalam</td>
</tr>
<tr>
<td>Mongolian</td>
<td>Mongolian</td>
</tr>
<tr>
<td>Oriya</td>
<td>Oriya</td>
</tr>
<tr>
<td>Sinhalese, Sinhala</td>
<td>Pali, Sinhalese</td>
</tr>
<tr>
<td>Syriac</td>
<td>Aramaic, Syriac</td>
</tr>
<tr>
<td>Tamil</td>
<td>Tamil</td>
</tr>
<tr>
<td>Telugu</td>
<td>Telugu</td>
</tr>
<tr>
<td>Thaana</td>
<td>Dhiveyi, Divehi, Maldivian</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language Group</th>
<th>Languages Include</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese</td>
<td>Cantonese, Mandarin, Pinyin phonograms</td>
</tr>
<tr>
<td>Japanese</td>
<td>Japanese (Hiragana, Kanji, Katakana)</td>
</tr>
</tbody>
</table>
Table D–2  (Cont.) Languages Supported by the World Lexer (Non-space-separated)

<table>
<thead>
<tr>
<th>Language Group</th>
<th>Languages Include</th>
</tr>
</thead>
<tbody>
<tr>
<td>Khmer</td>
<td>Cambodian, Khmer</td>
</tr>
<tr>
<td>Lao</td>
<td>Lao</td>
</tr>
<tr>
<td>Myanmar</td>
<td>Burmese</td>
</tr>
<tr>
<td>Thai</td>
<td>Thai</td>
</tr>
<tr>
<td>Tibetan</td>
<td>Dzongkha, Tibetan</td>
</tr>
</tbody>
</table>

Table D–3 shows languages not supported by the World Lexer.

Table D–3  Languages Not Supported by the World Lexer

<table>
<thead>
<tr>
<th>Language Group</th>
<th>Languages Include</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buhid</td>
<td>Buhid</td>
</tr>
<tr>
<td>Canadian Syllabics</td>
<td>Blackfoot, Carrier, Cree, Dakhelh, Inuit, Inuktitut, Naskapi, Nunavik, Nunavut, Ojibwe, Sayisi, Slavey</td>
</tr>
<tr>
<td>Cherokee</td>
<td>Cherokee</td>
</tr>
<tr>
<td>Cypriot</td>
<td>Cypriot</td>
</tr>
<tr>
<td>Limbu</td>
<td>Limbu</td>
</tr>
<tr>
<td>Oghem</td>
<td>Oghem</td>
</tr>
<tr>
<td>Runic</td>
<td>Runic</td>
</tr>
<tr>
<td>Tai Le (Tai Lu, Lue, Dai Le)</td>
<td>Tai Le</td>
</tr>
<tr>
<td>Ugaritic</td>
<td>Ugaritic</td>
</tr>
<tr>
<td>Yi</td>
<td>Yi</td>
</tr>
<tr>
<td>Yijang Hexagram</td>
<td>Yijang</td>
</tr>
</tbody>
</table>

Querying

Oracle Text supports the use of different query operators. Some operators can be set to behave in accordance with your language. This section summarizes the multilingual query features for these operators.
ABOUT Operator

Use the ABOUT operator to query on concepts. The system looks up concept information in the theme component of the index.

This feature is supported for English and French with CONTEXT indexes only.

Fuzzy Operator

This operator enables you to search for words that have similar spelling to specified word. Oracle Text supports fuzzy for English, German, Italian, Dutch, Spanish, Japanese, and Optical Character recognition (OCR).

Stem Operator

This operator enables you to search for words that have the same root as the specified term. For example, a stem of $sing expands into a query on the words sang, sung, sing. The Oracle Text stemmer supports the following languages: English, French, Spanish, Italian, German, Japanese and Dutch.

Supplied Stop Lists

A stoplist is a list of words that do not get indexed. These are usually common words in a language such as this, that, and can in English.

Oracle Text provides a default stoplist for English, Chinese (traditional and simplified), Danish, Dutch, Finnish, French, German, Italian, Portuguese, Spanish, and Swedish.

Knowledge Base

An Oracle Text knowledge base is a hierarchical tree of concepts used for theme indexing, ABOUT queries, and deriving themes for document services.

Oracle Text supplies knowledge bases in English and French only.

Knowledge Base Extension

You can extend theme functionality to languages other than English or French by loading your own knowledge base for any single byte white space delimited language, including Spanish.
Multi-Lingual Features Matrix

The following table summarizes the multilingual features for the supported languages.

Table D–4  Multilingual Features for Supported Languages

<table>
<thead>
<tr>
<th>LANGUAGE</th>
<th>BASE LETTER CONVERSION</th>
<th>ALTERNATE SPELLING</th>
<th>FUZZY MATCHING</th>
<th>LANGUAGE SPECIFIC LEXER</th>
<th>DEFAULT STOP LIST</th>
<th>STEMMING</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENGLISH</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GERMAN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JAPANESE</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>FRENCH</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SPANISH</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ITALIAN</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DUTCH</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PORTUGUESE</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>KOREAN</td>
<td>N/A</td>
<td>N/A</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>SIMPLIFIED CHINESE</td>
<td>N/A</td>
<td>N/A</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>TRADITIONAL CHINESE</td>
<td>N/A</td>
<td>N/A</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>DANISH</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SWEDISH</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>FINNISH</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
This appendix describes the default stoplists for all the different languages supported and list the stopwords in each. The following stoplists are described:

- English Default Stoplist
- Chinese Stoplist (Traditional)
- Chinese Stoplist (Simplified)
- Danish (dk) Default Stoplist
- Dutch (nl) Default Stoplist
- Finnish (sf) Default Stoplist
- French (f) Default Stoplist
- German (d) Default Stoplist
- Italian (i) Default Stoplist
- Portuguese (pt) Default Stoplist
- Spanish (e) Default Stoplist
- Swedish (s) Default Stoplist

**English Default Stoplist**

The following English words are defined as stop words:

<table>
<thead>
<tr>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>be</td>
<td>had</td>
<td>it</td>
<td>only</td>
<td>she</td>
<td>was</td>
</tr>
<tr>
<td>about</td>
<td>because</td>
<td>has</td>
<td>its</td>
<td>of</td>
<td>some</td>
<td>we</td>
</tr>
<tr>
<td>Stop word</td>
<td>Stop word</td>
<td>Stop word</td>
<td>Stop word</td>
<td>Stop word</td>
<td>Stop word</td>
<td>Stop word</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>after</td>
<td>been</td>
<td>have</td>
<td>last</td>
<td>on</td>
<td>such</td>
<td>were</td>
</tr>
<tr>
<td>all</td>
<td>but</td>
<td>he</td>
<td>more</td>
<td>one</td>
<td>than</td>
<td>when</td>
</tr>
<tr>
<td>also</td>
<td>by</td>
<td>her</td>
<td>most</td>
<td>or</td>
<td>that</td>
<td>which</td>
</tr>
<tr>
<td>an</td>
<td>can</td>
<td>his</td>
<td>mr</td>
<td>other</td>
<td>the</td>
<td>who</td>
</tr>
<tr>
<td>any</td>
<td>co</td>
<td>if</td>
<td>mrs</td>
<td>out</td>
<td>their</td>
<td>will</td>
</tr>
<tr>
<td>and</td>
<td>corp</td>
<td>in</td>
<td>ms</td>
<td>over</td>
<td>there</td>
<td>with</td>
</tr>
<tr>
<td>are</td>
<td>could</td>
<td>inc</td>
<td>mz</td>
<td>s</td>
<td>they</td>
<td>would</td>
</tr>
<tr>
<td>as</td>
<td>for</td>
<td>into</td>
<td>no</td>
<td>so</td>
<td>this</td>
<td>up</td>
</tr>
<tr>
<td>at</td>
<td>from</td>
<td>is</td>
<td>not</td>
<td>says</td>
<td>to</td>
<td></td>
</tr>
</tbody>
</table>

**Chinese Stoplist (Traditional)**

The following traditional Chinese words are defined in the default stoplist for this language.

<table>
<thead>
<tr>
<th>目前</th>
<th>由於</th>
<th>因此</th>
<th>他們</th>
<th>可能</th>
<th>沒有</th>
<th>希望</th>
</tr>
</thead>
<tbody>
<tr>
<td>有關</td>
<td>不過</td>
<td>可以</td>
<td>如果</td>
<td>對於</td>
<td>因為</td>
<td>是否</td>
</tr>
<tr>
<td>但是</td>
<td>相當</td>
<td>其中</td>
<td>其他</td>
<td>雖然</td>
<td>我們</td>
<td>包括</td>
</tr>
<tr>
<td>必須</td>
<td>以上</td>
<td>之後</td>
<td>所以</td>
<td>以及</td>
<td>許多</td>
<td>最近</td>
</tr>
<tr>
<td>至於</td>
<td>一般</td>
<td>不是</td>
<td>不能</td>
<td>而且</td>
<td>引起</td>
<td>如何</td>
</tr>
<tr>
<td>除了</td>
<td>少</td>
<td>最後</td>
<td>就是</td>
<td>分別</td>
<td>加強</td>
<td>甚至</td>
</tr>
<tr>
<td>繼續</td>
<td>另外</td>
<td>共同</td>
<td>只有</td>
<td>了解</td>
<td>根據</td>
<td>已經</td>
</tr>
<tr>
<td>過去</td>
<td>所有</td>
<td>不會</td>
<td>以來</td>
<td>任何</td>
<td>一直</td>
<td>不同</td>
</tr>
<tr>
<td>立即</td>
<td>左右</td>
<td>經過</td>
<td>尤其</td>
<td>使得</td>
<td>相關</td>
<td>當時</td>
</tr>
<tr>
<td>進入</td>
<td>並不</td>
<td>據了解</td>
<td>現在</td>
<td>只是</td>
<td>需要</td>
<td>原因</td>
</tr>
<tr>
<td>只要</td>
<td>否則</td>
<td>並未</td>
<td>什麼</td>
<td>如此</td>
<td>不要</td>
<td></td>
</tr>
</tbody>
</table>

**Chinese Stoplist (Simplified)**

The following simplified Chinese words are defined in the default stoplist for this language.
Danish (dk) Default Stoplist

The following Danish words are defined in the default stoplist for this language:

<table>
<thead>
<tr>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
</tr>
</thead>
<tbody>
<tr>
<td>af</td>
<td>en</td>
<td>god</td>
<td>hvordan</td>
<td>med</td>
<td>og</td>
<td>udenfor</td>
</tr>
<tr>
<td>aldrig</td>
<td>et</td>
<td>han</td>
<td>I</td>
<td>meget</td>
<td>oppe</td>
<td>under</td>
</tr>
<tr>
<td>alle</td>
<td>endnu</td>
<td>her</td>
<td>De</td>
<td>mellem</td>
<td>på</td>
<td>ved</td>
</tr>
<tr>
<td>altid</td>
<td>få</td>
<td>hos</td>
<td>i</td>
<td>mere</td>
<td>rask</td>
<td>vi</td>
</tr>
<tr>
<td>bagved</td>
<td>lidt</td>
<td>hosfor</td>
<td>imod</td>
<td>mindre</td>
<td>hurtig</td>
<td></td>
</tr>
<tr>
<td>de</td>
<td>fjern</td>
<td>hun</td>
<td>ja</td>
<td>når</td>
<td>sammen</td>
<td></td>
</tr>
<tr>
<td>der</td>
<td>for</td>
<td>hvad</td>
<td>jeg</td>
<td>hvonår</td>
<td>temmelig</td>
<td></td>
</tr>
<tr>
<td>du</td>
<td>foran</td>
<td>hvem</td>
<td>langsom</td>
<td>nede</td>
<td>nok</td>
<td></td>
</tr>
<tr>
<td>efter</td>
<td>fra</td>
<td>hvor</td>
<td>mange</td>
<td>nej</td>
<td>til</td>
<td></td>
</tr>
<tr>
<td>eller</td>
<td>gennem</td>
<td>hvorhen</td>
<td>måske</td>
<td>nu</td>
<td>uden</td>
<td></td>
</tr>
</tbody>
</table>

Dutch (nl) Default Stoplist

The following Dutch words are defined in the default stoplist for this language:

<table>
<thead>
<tr>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
</tr>
</thead>
<tbody>
<tr>
<td>aan</td>
<td>betreffende</td>
<td>eer</td>
<td>had</td>
<td>juist</td>
<td>na</td>
<td>overeind</td>
<td>van</td>
<td>weer</td>
</tr>
</tbody>
</table>

Supplied Stoplists  E-3
### Finnish (sf) Default Stoplist

The following Finnish words are defined in the default stoplist for this language:

<table>
<thead>
<tr>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
</tr>
</thead>
<tbody>
<tr>
<td>aangaande</td>
<td>bij</td>
<td>eerdat</td>
<td>hadden</td>
<td>jullie</td>
<td>naar</td>
<td>overigens</td>
<td>vandaan</td>
<td>weg</td>
</tr>
<tr>
<td>aangezien</td>
<td>binnen</td>
<td>eerder</td>
<td>hare</td>
<td>kan</td>
<td>nadat</td>
<td>pas</td>
<td>vanuit</td>
<td>wengens</td>
</tr>
<tr>
<td>achter</td>
<td>binnenin</td>
<td>eerlang</td>
<td>heb</td>
<td>klaar</td>
<td>net</td>
<td>precies</td>
<td>vanwege</td>
<td>wel</td>
</tr>
<tr>
<td>achterna</td>
<td>boven</td>
<td>eerst</td>
<td>hebben</td>
<td>kon</td>
<td>niet</td>
<td>reeds</td>
<td>veeleer</td>
<td>weldra</td>
</tr>
<tr>
<td>afgelopen</td>
<td>bovenal</td>
<td>elk</td>
<td>hebt</td>
<td>konden</td>
<td>noch</td>
<td>rond</td>
<td>verder</td>
<td>welk</td>
</tr>
<tr>
<td>al</td>
<td>bovendien</td>
<td>elke</td>
<td>heeft</td>
<td>krachtens</td>
<td>nog</td>
<td>rondom</td>
<td>vervolgens</td>
<td>welke</td>
</tr>
<tr>
<td>aldaar</td>
<td>bovengenoemd</td>
<td>en</td>
<td>hem</td>
<td>kunnen</td>
<td>nogal</td>
<td>sedert</td>
<td>vol</td>
<td>wie</td>
</tr>
<tr>
<td>als dus</td>
<td>bovenstaand</td>
<td>enig</td>
<td>hen</td>
<td>kunt</td>
<td>nu</td>
<td>sinds</td>
<td>volgens</td>
<td>wiens</td>
</tr>
<tr>
<td>alhoewel</td>
<td>bovenvermeld</td>
<td>enigszins</td>
<td>het</td>
<td>later</td>
<td>of</td>
<td>sindsdien</td>
<td>voor</td>
<td>wier</td>
</tr>
<tr>
<td>alias</td>
<td>buiten</td>
<td>enkel</td>
<td>hierbeneden</td>
<td>liever</td>
<td>ofschoon</td>
<td>slechts</td>
<td>vooraf</td>
<td>wij</td>
</tr>
<tr>
<td>alle</td>
<td>daar</td>
<td>er</td>
<td>hierboven</td>
<td>maar</td>
<td>om</td>
<td>sommige</td>
<td>vooral</td>
<td>wijzelf</td>
</tr>
<tr>
<td>allebei</td>
<td>daarheen</td>
<td>erdoor</td>
<td>hij</td>
<td>mag</td>
<td>omdat</td>
<td>spoedig</td>
<td>vooralsnog</td>
<td>zal</td>
</tr>
<tr>
<td>alleen</td>
<td>daarin</td>
<td>even</td>
<td>hoe</td>
<td>meer</td>
<td>omhoog</td>
<td>steeds</td>
<td>voorbij</td>
<td>ze</td>
</tr>
<tr>
<td>als nog</td>
<td>daarna</td>
<td>eveneens</td>
<td>hoewel</td>
<td>met</td>
<td>omlaag</td>
<td>tamelijk</td>
<td>voordat</td>
<td>zelfs</td>
</tr>
<tr>
<td>altijd</td>
<td>daarmet</td>
<td>evenwel</td>
<td>hun</td>
<td>mezelf</td>
<td>omstreeks</td>
<td>tensij</td>
<td>voordezen</td>
<td>zichzelf</td>
</tr>
<tr>
<td>altoos</td>
<td>daarom</td>
<td>gauw</td>
<td>hunne</td>
<td>mij</td>
<td>omtrent</td>
<td>terwijl</td>
<td>voordien</td>
<td>zij</td>
</tr>
<tr>
<td>ander</td>
<td>daarop</td>
<td>gedurende</td>
<td>ik</td>
<td>mijn</td>
<td>omver</td>
<td>thans</td>
<td>voorheen</td>
<td>zijn</td>
</tr>
<tr>
<td>andere</td>
<td>daarvanlangs</td>
<td>geen</td>
<td>ikzelf</td>
<td>mijnent</td>
<td>onder</td>
<td>tijdens</td>
<td>voorop</td>
<td>zijne</td>
</tr>
<tr>
<td>anders</td>
<td>dan</td>
<td>gehad</td>
<td>in</td>
<td>mijner</td>
<td>ondertussen</td>
<td>toch</td>
<td>vooruit</td>
<td>zo</td>
</tr>
<tr>
<td>anderszins</td>
<td>dat</td>
<td>gekund</td>
<td>inmiddels</td>
<td>mijzelf</td>
<td>ongeveer</td>
<td>toen</td>
<td>vrij</td>
<td>zodra</td>
</tr>
<tr>
<td>behalve</td>
<td>de</td>
<td>geleden</td>
<td>inzake</td>
<td>misschien</td>
<td>ons</td>
<td>toenmaals</td>
<td>vroeg</td>
<td>zonder</td>
</tr>
<tr>
<td>behoudens</td>
<td>die</td>
<td>ghelijk</td>
<td>is</td>
<td>mocht</td>
<td>onszelf</td>
<td>toenmalig</td>
<td>waar</td>
<td>zou</td>
</tr>
<tr>
<td>beide</td>
<td>dikwijk</td>
<td>gemoeten</td>
<td>jezelf</td>
<td>mochten</td>
<td>onze</td>
<td>tot</td>
<td>waarom</td>
<td>zouden</td>
</tr>
<tr>
<td>beiden</td>
<td>dit</td>
<td>gemogen</td>
<td>jij</td>
<td>moest</td>
<td>ook</td>
<td>totdat</td>
<td>wanneer</td>
<td>zowat</td>
</tr>
<tr>
<td>ben</td>
<td>door</td>
<td>geweest</td>
<td>jizelf</td>
<td>moesten</td>
<td>op</td>
<td>tussen</td>
<td>want</td>
<td>zulke</td>
</tr>
<tr>
<td>beneden</td>
<td>doorgaand</td>
<td>gewoon</td>
<td>jou</td>
<td>moet</td>
<td>opnieuw</td>
<td>uit</td>
<td>waren</td>
<td>zullen</td>
</tr>
<tr>
<td>bent</td>
<td>dus</td>
<td>gewoonweg</td>
<td>jouw</td>
<td>moeten</td>
<td>opzij</td>
<td>uitgezonderd</td>
<td>was</td>
<td>zult</td>
</tr>
<tr>
<td>bepaald</td>
<td>echter</td>
<td>haar</td>
<td>jouwe</td>
<td>mogen</td>
<td>over</td>
<td>vaak</td>
<td>wat</td>
<td></td>
</tr>
</tbody>
</table>
### French (f) Default Stoplist

The following French words are defined in the default stoplist for this language:

<table>
<thead>
<tr>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
</tr>
</thead>
<tbody>
<tr>
<td>aina</td>
<td>hyvin</td>
<td>kesken</td>
<td>me</td>
<td>nyt</td>
<td>takia</td>
<td>yhdessä</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>alla</td>
<td>hoikein</td>
<td>kukka</td>
<td>mikä</td>
<td>oikea</td>
<td>tässä</td>
<td>ylöös</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ansiosta</td>
<td>ilman</td>
<td>kyllä</td>
<td>miksi</td>
<td>oikealla</td>
<td>te</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ei</td>
<td>ja</td>
<td>kylliksi</td>
<td>milloin</td>
<td>paljon</td>
<td>ulkopuolella</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>enemmän</td>
<td>jälkeen</td>
<td>tarpeksi</td>
<td>milloinkan</td>
<td>siellä</td>
<td>vähän</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ennen</td>
<td>jos</td>
<td>lähellä</td>
<td>koskaan</td>
<td>sinä</td>
<td>vahemmän</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>etessa</td>
<td>kanssa</td>
<td>läpi</td>
<td>minä</td>
<td>ssa</td>
<td>vasen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>haikki</td>
<td>kaukana</td>
<td>liian</td>
<td>missä</td>
<td>sta</td>
<td>vasenmalla</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hän</td>
<td>kenties</td>
<td>lla</td>
<td>miten</td>
<td>suoraan</td>
<td>vastan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>he</td>
<td>ehkä</td>
<td>luona</td>
<td>kuinkan</td>
<td>tai</td>
<td>vielä</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hitaasti</td>
<td>keskellä</td>
<td>lla</td>
<td>nopeasti</td>
<td>takana</td>
<td>vieressä</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Supplied Stoplists**

E-5
**German (d) Default Stoplist**

The following German words are defined in the default stoplist for this language:

<table>
<thead>
<tr>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
</tr>
</thead>
<tbody>
<tr>
<td>aussitôt</td>
<td>cette</td>
<td>donné</td>
<td>jadis</td>
<td>mes</td>
<td>où</td>
<td>quiconque</td>
<td>telle</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>autant</td>
<td>ceux</td>
<td>dont</td>
<td>je</td>
<td>mien</td>
<td>par</td>
<td>quoi</td>
<td>telles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>autour</td>
<td>chacun</td>
<td>du</td>
<td>jusqu</td>
<td>mienne</td>
<td>parce</td>
<td>quoique</td>
<td>tes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aux</td>
<td>chacune</td>
<td>duquel</td>
<td>jusque</td>
<td>miennes</td>
<td>parmi</td>
<td>sa</td>
<td>tienne</td>
<td></td>
<td></td>
</tr>
<tr>
<td>auxquelles</td>
<td>chaque</td>
<td>durant</td>
<td>la</td>
<td>miens</td>
<td>plus</td>
<td>sans</td>
<td>tiennes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>auxquels</td>
<td>chez</td>
<td>elle</td>
<td>laquelle</td>
<td>moins</td>
<td>plusieurs</td>
<td>sauf</td>
<td>tiens</td>
<td></td>
<td></td>
</tr>
<tr>
<td>avec</td>
<td>combien</td>
<td>elles</td>
<td>là</td>
<td>moment</td>
<td>pour</td>
<td>se</td>
<td>toi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>à</td>
<td>comme</td>
<td>en</td>
<td>le</td>
<td>mon</td>
<td>pourquoi</td>
<td>selon</td>
<td>ton</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Stop word**

<table>
<thead>
<tr>
<th>ab</th>
<th>dann</th>
<th>des</th>
<th>es</th>
<th>ihnen</th>
<th>keinem</th>
<th>obgleich</th>
<th>sondern</th>
<th>welchem</th>
</tr>
</thead>
<tbody>
<tr>
<td>aber</td>
<td>daran</td>
<td>desselben</td>
<td>etwa</td>
<td>ihr</td>
<td>keinen</td>
<td>oder</td>
<td>sonst</td>
<td>welchen</td>
</tr>
<tr>
<td>allein</td>
<td>darauf</td>
<td>dessen</td>
<td>etwas</td>
<td>ihre</td>
<td>keiner</td>
<td>ohne</td>
<td>soviel</td>
<td>welcher</td>
</tr>
<tr>
<td>als</td>
<td>daraus</td>
<td>dich</td>
<td>euch</td>
<td>ihre</td>
<td>keines</td>
<td>paar</td>
<td>soweit</td>
<td>welches</td>
</tr>
<tr>
<td>also</td>
<td>darin</td>
<td>die</td>
<td>euer</td>
<td>ihrem</td>
<td>man</td>
<td>sehr</td>
<td>über</td>
<td>wem</td>
</tr>
<tr>
<td>am</td>
<td>darüber</td>
<td>dies</td>
<td>eure</td>
<td>ihrem</td>
<td>mehr</td>
<td>sei</td>
<td>um</td>
<td>wen</td>
</tr>
<tr>
<td>an</td>
<td>darum</td>
<td>diese</td>
<td>eurem</td>
<td>ihren</td>
<td>mein</td>
<td>sein</td>
<td>und</td>
<td>wenn</td>
</tr>
<tr>
<td>auch</td>
<td>darunter</td>
<td>dieselbe</td>
<td>euren</td>
<td>Ihren</td>
<td>meine</td>
<td>seine</td>
<td>uns</td>
<td>wer</td>
</tr>
<tr>
<td>auf</td>
<td>das</td>
<td>dieselben</td>
<td>eurer</td>
<td>Ihrem</td>
<td>meinem</td>
<td>seinem</td>
<td>unser</td>
<td>weshalb</td>
</tr>
<tr>
<td>aus</td>
<td>dasselbe</td>
<td>diesem</td>
<td>eures</td>
<td>ihres</td>
<td>meinen</td>
<td>seinen</td>
<td>unsre</td>
<td>wessen</td>
</tr>
<tr>
<td>außer</td>
<td>daß</td>
<td>diesen</td>
<td>für</td>
<td>ihres</td>
<td>meiner</td>
<td>seiner</td>
<td>unsrem</td>
<td>wie</td>
</tr>
<tr>
<td>bald</td>
<td>davon</td>
<td>dieser</td>
<td>fürs</td>
<td>Ihres</td>
<td>meines</td>
<td>seines</td>
<td>unsren</td>
<td>wir</td>
</tr>
<tr>
<td>bei</td>
<td>davor</td>
<td>dieses</td>
<td>ganz</td>
<td>im</td>
<td>mich</td>
<td>seit</td>
<td>unsrer</td>
<td>wo</td>
</tr>
<tr>
<td>beim</td>
<td>dazu</td>
<td>dir</td>
<td>gar</td>
<td>in</td>
<td>mir</td>
<td>seitdem</td>
<td>unsres</td>
<td>womit</td>
</tr>
<tr>
<td>bin</td>
<td>dazwischen</td>
<td>doch</td>
<td>gegen</td>
<td>ist</td>
<td>mit</td>
<td>selbst</td>
<td>vom</td>
<td>zu</td>
</tr>
<tr>
<td>bis</td>
<td>dein</td>
<td>dort</td>
<td>genau</td>
<td>ja</td>
<td>nach</td>
<td>sich</td>
<td>von</td>
<td>zum</td>
</tr>
<tr>
<td>bißchen</td>
<td>deine</td>
<td>du</td>
<td>gewesen</td>
<td>je</td>
<td>nachdem</td>
<td>Sie</td>
<td>vor</td>
<td>zur</td>
</tr>
<tr>
<td>bist</td>
<td>demen</td>
<td>ebenso</td>
<td>her</td>
<td>jedesmal</td>
<td>nämlich</td>
<td>sie</td>
<td>während</td>
<td>zwar</td>
</tr>
</tbody>
</table>
### Italian (i) Default Stoplist

The following Italian words are defined in the default stoplist for this language:

<table>
<thead>
<tr>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
</tr>
</thead>
<tbody>
<tr>
<td>da</td>
<td>deinen</td>
<td>ehe</td>
<td>herein</td>
<td>jedoch</td>
<td>neben</td>
<td>sind</td>
<td>war</td>
<td>zwischen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dabei</td>
<td>deiner</td>
<td>ein</td>
<td>herum</td>
<td>jene</td>
<td>nein</td>
<td>so</td>
<td>wieder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dadurch</td>
<td>deines</td>
<td>eine</td>
<td>hin</td>
<td>jenem</td>
<td>nicht</td>
<td>sogar</td>
<td>wären</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dafür</td>
<td>dem</td>
<td>einem</td>
<td>hinter</td>
<td>jenen</td>
<td>nichts</td>
<td>solch</td>
<td>warum</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dagegen</td>
<td>demselben</td>
<td>einen</td>
<td>hintern</td>
<td>jener</td>
<td>noch</td>
<td>solche</td>
<td>was</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dahinter</td>
<td>den</td>
<td>einer</td>
<td>ich</td>
<td>jenes</td>
<td>nun</td>
<td>solchem</td>
<td>wegen</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>damit</td>
<td>denn</td>
<td>eines</td>
<td>ihm</td>
<td>kaum</td>
<td>nur</td>
<td>solchen</td>
<td>weil</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>danach</td>
<td>der</td>
<td>entlang</td>
<td>ihn</td>
<td>kein</td>
<td>ob</td>
<td>solcher</td>
<td>weit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>daneben</td>
<td>derselben</td>
<td>er</td>
<td>Ihnen</td>
<td>keine</td>
<td>ober</td>
<td>solches</td>
<td>welche</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Supply Stoplists**
Portuguese (pt) Default Stoplist

The following Portuguese words are defined in the default stoplist for this language:

<table>
<thead>
<tr>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
</tr>
</thead>
<tbody>
<tr>
<td>comunque</td>
<td>di</td>
<td>io</td>
<td>noi</td>
<td>se</td>
<td>tu</td>
<td></td>
</tr>
<tr>
<td>con</td>
<td>dopo</td>
<td>l’</td>
<td>nonché</td>
<td>sebbene</td>
<td>tuo</td>
<td></td>
</tr>
<tr>
<td>contro</td>
<td>dove</td>
<td>la</td>
<td>nondimeno</td>
<td>sennonché</td>
<td>tuttavía</td>
<td></td>
</tr>
<tr>
<td>cosa</td>
<td>dunque</td>
<td>le</td>
<td>nostro</td>
<td>senza</td>
<td>tutti</td>
<td></td>
</tr>
</tbody>
</table>

Spanish (e) Default Stoplist

The following Spanish words are defined in the default stoplist for this language:

<table>
<thead>
<tr>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>bem</td>
<td>e</td>
<td>longe</td>
<td>para</td>
<td>se</td>
<td>você</td>
<td></td>
</tr>
<tr>
<td>abaixo</td>
<td>com</td>
<td>ela</td>
<td>mais</td>
<td>por</td>
<td>sem</td>
<td>vocês</td>
<td></td>
</tr>
<tr>
<td>adiante</td>
<td>como</td>
<td>elas</td>
<td>menos</td>
<td>porque</td>
<td>sempre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>agora</td>
<td>contra</td>
<td>élé</td>
<td>mucho</td>
<td>poco</td>
<td>sim</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ali</td>
<td>debaixo</td>
<td>eles</td>
<td>não</td>
<td>próximo</td>
<td>sob</td>
<td></td>
<td></td>
</tr>
<tr>
<td>antes</td>
<td>demais</td>
<td>em</td>
<td>ninguem</td>
<td>qual</td>
<td>sobre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aqui</td>
<td>depois</td>
<td>entre</td>
<td>nós</td>
<td>quando</td>
<td>talvez</td>
<td></td>
<td></td>
</tr>
<tr>
<td>até</td>
<td>depressa</td>
<td>eu</td>
<td>nunca</td>
<td>quanto</td>
<td>todas</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atras</td>
<td>devagar</td>
<td>fora</td>
<td>onde</td>
<td>que</td>
<td>todos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bastante</td>
<td>direito</td>
<td>junto</td>
<td>ou</td>
<td>quem</td>
<td>vagarosamente</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Swedish (s) Default Stoplist

The following Swedish words are defined in the default stoplist for this language:

<table>
<thead>
<tr>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
</tr>
</thead>
<tbody>
<tr>
<td>ajenos</td>
<td>como</td>
<td>del</td>
<td>hasta</td>
<td>muchísimas</td>
<td>nunca</td>
<td>quién</td>
<td>te</td>
<td>vuestra</td>
</tr>
<tr>
<td>al</td>
<td>cómo</td>
<td>demasiada</td>
<td>jamás</td>
<td>muchísimas</td>
<td>os</td>
<td>ser</td>
<td>tener</td>
<td>vuestras</td>
</tr>
<tr>
<td>algo</td>
<td>con</td>
<td>demasiadas</td>
<td>junto</td>
<td>muchísimo</td>
<td>otra</td>
<td>sí</td>
<td>ti</td>
<td>vuestro</td>
</tr>
<tr>
<td>alguna</td>
<td>conmigo</td>
<td>demasiado</td>
<td>juntos</td>
<td>muchísimos</td>
<td>otras</td>
<td>siempre</td>
<td>toda</td>
<td>vuestros</td>
</tr>
<tr>
<td>algunas</td>
<td>consigo</td>
<td>demasiados</td>
<td>la</td>
<td>mucho</td>
<td>otro</td>
<td>sí</td>
<td>todas</td>
<td>y</td>
</tr>
<tr>
<td>alguno</td>
<td>contigo</td>
<td>demás</td>
<td>las</td>
<td>muchos</td>
<td>otros</td>
<td>sin</td>
<td>todo</td>
<td>yo</td>
</tr>
<tr>
<td>algunos</td>
<td>cualquier</td>
<td>el</td>
<td>lo</td>
<td>muy</td>
<td>para</td>
<td>Sr</td>
<td>todos</td>
<td></td>
</tr>
<tr>
<td>algún</td>
<td>cualquiera</td>
<td>ella</td>
<td>los</td>
<td>nada</td>
<td>parecer</td>
<td>Sra</td>
<td>tomar</td>
<td></td>
</tr>
<tr>
<td>allá</td>
<td>cualesquieras</td>
<td>ellas</td>
<td>mas</td>
<td>ni</td>
<td>poca</td>
<td>Sres</td>
<td>tuya</td>
<td></td>
</tr>
<tr>
<td>allí</td>
<td>cuan</td>
<td>ellos</td>
<td>más</td>
<td>ninguna</td>
<td>pocas</td>
<td>Sta</td>
<td>tuyo</td>
<td></td>
</tr>
<tr>
<td>aquel</td>
<td>cuanta</td>
<td>el</td>
<td>me</td>
<td>ningunas</td>
<td>poco</td>
<td>suya</td>
<td>tú</td>
<td></td>
</tr>
<tr>
<td>aquella</td>
<td>cuantas</td>
<td>esa</td>
<td>menos</td>
<td>ninguno</td>
<td>pocos</td>
<td>suyas</td>
<td>un</td>
<td></td>
</tr>
<tr>
<td>aquellas</td>
<td>cuánta</td>
<td>esas</td>
<td>mía</td>
<td>ningunos</td>
<td>por</td>
<td>suyo</td>
<td>una</td>
<td></td>
</tr>
<tr>
<td>aquello</td>
<td>cuántas</td>
<td>ese</td>
<td>mientras</td>
<td>no</td>
<td>porque</td>
<td>suyos</td>
<td>unas</td>
<td></td>
</tr>
<tr>
<td>aquellos</td>
<td>cuanto</td>
<td>esos</td>
<td>mío</td>
<td>nos</td>
<td>que</td>
<td>tal</td>
<td>unos</td>
<td></td>
</tr>
</tbody>
</table>

### Supplied Stoplists  E-9
### Swedish (s) Default Stoplist

<table>
<thead>
<tr>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
<th>Stop word</th>
</tr>
</thead>
<tbody>
<tr>
<td>avser</td>
<td>framför</td>
<td>mindre</td>
<td>utom</td>
</tr>
<tr>
<td>avses</td>
<td>från</td>
<td>mot</td>
<td>vad</td>
</tr>
<tr>
<td>bakom</td>
<td>genom</td>
<td>myckett</td>
<td>väl</td>
</tr>
<tr>
<td>bra</td>
<td>gott</td>
<td>när</td>
<td>var</td>
</tr>
<tr>
<td>bredvid</td>
<td>hamske</td>
<td>nära</td>
<td>varför</td>
</tr>
<tr>
<td>dä</td>
<td>han</td>
<td>nej</td>
<td>vart</td>
</tr>
<tr>
<td>där</td>
<td>här</td>
<td>nere</td>
<td>varthän</td>
</tr>
<tr>
<td>de</td>
<td>hellre</td>
<td>ni</td>
<td>vem</td>
</tr>
<tr>
<td>dem</td>
<td>hon</td>
<td>nu</td>
<td>vems</td>
</tr>
<tr>
<td>den</td>
<td>hos</td>
<td>och</td>
<td>vi</td>
</tr>
<tr>
<td>denna</td>
<td>hur</td>
<td>oksa</td>
<td>vid</td>
</tr>
<tr>
<td>deras</td>
<td>i</td>
<td>om</td>
<td>vilken</td>
</tr>
<tr>
<td>dess</td>
<td>in</td>
<td>över</td>
<td></td>
</tr>
<tr>
<td>det</td>
<td>ingen</td>
<td>på</td>
<td></td>
</tr>
<tr>
<td>detta</td>
<td>innan</td>
<td>så</td>
<td></td>
</tr>
<tr>
<td>du</td>
<td>inte</td>
<td>sådan</td>
<td></td>
</tr>
</tbody>
</table>
This appendix describes the scoring algorithm for word queries. You obtain score using the SCORE operator.

Note: This appendix discusses how Oracle Text 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 Text uses an inverse frequency algorithm based on Salton's formula.

Inverse frequency scoring assumes that frequently occurring terms in a document set are noise terms, and so these terms are scored lower. For a document to score high, the query term must occur frequently in the document but infrequently in the document set as a whole.

The following table illustrates Oracle Text’s inverse frequency scoring. The first column shows the number of documents in the document set, and the second column shows the number of terms in the document necessary to score 100.

This table assumes that only one document in the set contains the query term.

<table>
<thead>
<tr>
<th>Number of Documents in Document Set</th>
<th>Occurrences of Term in Document Needed to Score 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
</tr>
</tbody>
</table>
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 Database administrator must optimize the index. Perfect relevance ranking is obtained by executing a query right after optimizing the index.

If DML is light, Oracle Database still gives fairly accurate relevance ranking.

In either case, you or your Oracle Database administrator must synchronize the index with CTX_DDL.SYNC_INDEX.
This appendix lists all of the views provided by Oracle Text. The system provides the following views:

- `CTX_CLASSES`
- `CTX_INDEXES`
- `CTX_INDEX_ERRORS`
- `CTX_INDEX_OBJECTS`
- `CTX_INDEX_PARTITIONS`
- `CTX_INDEX_SETS`
- `CTX_INDEX_SET_INDEXES`
- `CTX_INDEX_SUB_LEXERS`
- `CTX_INDEX_SUB_LEXER_VALUES`
- `CTX_INDEX_VALUES`
- `CTX_OBJECTS`
- `CTX_OBJECT_ATTRIBUTES`
- `CTX_OBJECT_ATTRIBUTE_LOV`
- `CTX_PARAMETERS`
- `CTX_PENDING`
- `CTX_PREFERENCES`
- `CTX_PREFERENCE_VALUES`
- `CTX_SECTIONS`
- CTX_SECTION_GROUPS
- CTX_SQES
- CTX_STOPLISTS
- CTX_STOPWORDS
- CTX_SUB_LEXERS
- CTX_THESAURI
- CTX_THES_PHRASES
- CTX_TRACE_VALUES
- CTX_USER_INDEXES
- CTX_USER_INDEX_ERRORS
- CTX_USER_INDEX_OBJECTS
- CTX_USER_INDEX_PARTITIONS
- CTX_USER_INDEX_SETS
- CTX_USER_INDEX_SET_INDEXES
- CTX_USER_INDEX_SUB_LEXERS
- CTX_USER_INDEX_SUB_LEXER_VALS
- CTX_USER_INDEX_VALUES
- CTX_USER_PENDING
- CTX_USER_PREFERENCES
- CTX_USER_PREFERENCE_VALUES
- CTX_USER_SECTIONS
- CTX_USER_SECTION_GROUPS
- CTX_USER_SQES
- CTX_USER_STOPLISTS
- CTX_USER_STOPWORDS
- CTX_USER_SUB_LEXERS
- CTX_USER_THESAURI
- CTX_USER_THES_PHRASES
This view displays all the preference categories registered in the Text data dictionary. It can be queried by any user.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA_NAME</td>
<td>VARCHAR2(30)</td>
<td>Class name</td>
</tr>
<tr>
<td>CLA_DESCRIPTION</td>
<td>VARCHAR2(80)</td>
<td>Class description</td>
</tr>
</tbody>
</table>

This view displays all indexes that are registered in the Text data dictionary for the current user. It can be queried by CTXSYS.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDX_CHARSET_COLUMN</td>
<td>VARCHAR2(256)</td>
<td>Name of the charset column in base table.</td>
</tr>
<tr>
<td>IDX_DOCID_COUNT</td>
<td>NUMBER</td>
<td>Number of documents indexed.</td>
</tr>
<tr>
<td>IDX_FORMAT_COLUMN</td>
<td>VARCHAR2(256)</td>
<td>Name of the format column in base table.</td>
</tr>
<tr>
<td>IDX_KEY_NAME</td>
<td>VARCHAR2(256)</td>
<td>Primary key column(s).</td>
</tr>
<tr>
<td>IDX_ID</td>
<td>NUMBER</td>
<td>Internal index id.</td>
</tr>
<tr>
<td>IDX_LANGUAGE_COLUMN</td>
<td>VARCHAR2(256)</td>
<td>Name of the language column in base table.</td>
</tr>
<tr>
<td>IDX_NAME</td>
<td>VARCHAR2(30)</td>
<td>Name of index.</td>
</tr>
<tr>
<td>IDX_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Owner of index.</td>
</tr>
<tr>
<td>IDX_STATUS</td>
<td>VARCHAR2(12)</td>
<td>Status.</td>
</tr>
<tr>
<td>IDX_SYNC_TYPE</td>
<td>VARCHAR2(20)</td>
<td>Type of synching: MANUAL, AUTOMATIC, or ON COMMIT.</td>
</tr>
<tr>
<td>IDX_TABLE</td>
<td>VARCHAR2(30)</td>
<td>Table name.</td>
</tr>
<tr>
<td>IDX_TABLE_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Owner of table.</td>
</tr>
<tr>
<td>IDX_TEXT_NAME</td>
<td>VARCHAR2(30)</td>
<td>Text column name.</td>
</tr>
</tbody>
</table>
**CTX_INDEX_ERRORS**

This view displays the DML errors and is queryable by CTXSYS.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR_INDEX_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>Index owner.</td>
</tr>
<tr>
<td>ERR_INDEX_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Name of index.</td>
</tr>
<tr>
<td>ERR_TIMESTAMP</td>
<td>DATE</td>
<td>Time of error.</td>
</tr>
<tr>
<td>ERR_TEXTKEY</td>
<td>VARCHAR2 (18)</td>
<td>ROWID of errored document or name of errored operation (for example, ALTER INDEX)</td>
</tr>
<tr>
<td>ERR_TEXT</td>
<td>VARCHAR2 (4000)</td>
<td>Error text.</td>
</tr>
</tbody>
</table>

**CTX_INDEX_OBJECTS**

This view displays the objects that are used for each class in the index. It can be queried by CTXSYS.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXO_INDEX_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>Index owner.</td>
</tr>
<tr>
<td>IXO_INDEX_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Index name.</td>
</tr>
<tr>
<td>IXO_CLASS</td>
<td>VARCHAR2 (30)</td>
<td>Class name.</td>
</tr>
<tr>
<td>IXO_OBJECT</td>
<td>VARCHAR2 (30)</td>
<td>Object name.</td>
</tr>
</tbody>
</table>

**CTX_INDEX_PARTITIONS**

This view displays all index partitions. It can be queried by CTXSYS.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXP_ID</td>
<td>NUMBER (38)</td>
<td>Index partition id.</td>
</tr>
<tr>
<td>IXP_INDEX_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>Index owner.</td>
</tr>
<tr>
<td>IXP_INDEX_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Index name.</td>
</tr>
<tr>
<td>IXP_INDEX_PARTITION_</td>
<td>VARCHAR2 (30)</td>
<td>Index partition name.</td>
</tr>
</tbody>
</table>
CTX_INDEX_SUB_LEXERS

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXP_SYNC_TYPE</td>
<td>VARCHAR2(20)</td>
<td>Type of synching: MANUAL, AUTOMATIC, or ON COMMIT.</td>
</tr>
<tr>
<td>IXP_TABLE_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Table owner.</td>
</tr>
<tr>
<td>IXP_TABLE_NAME</td>
<td>VARCHAR2(30)</td>
<td>Table name.</td>
</tr>
<tr>
<td>IXP_TABLE_PARTITION_NAME</td>
<td>VARCHAR2(30)</td>
<td>Table partition name.</td>
</tr>
<tr>
<td>IXP_DOCID_COUNT</td>
<td>NUMBER(38)</td>
<td>Number of documents associated with the partition.</td>
</tr>
<tr>
<td>IXP_STATUS</td>
<td>VARCHAR2(12)</td>
<td>Partition status.</td>
</tr>
</tbody>
</table>

CTX_INDEX_SETS

This view displays all index set names. It can be queried by any user.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXS_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Index set owner.</td>
</tr>
<tr>
<td>IXS_NAME</td>
<td>VARCHAR2(30)</td>
<td>Index set name.</td>
</tr>
</tbody>
</table>

CTX_INDEX_SET_INDEXES

This view displays all the sub-indexes in an index set. It can be queried by any user.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXX_INDEX_SET_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Index set owner.</td>
</tr>
<tr>
<td>IXX_INDEX_SET_NAME</td>
<td>VARCHAR2(30)</td>
<td>Index set name.</td>
</tr>
<tr>
<td>IXX_COLLIST</td>
<td>VARCHAR2(500)</td>
<td>Column list of the sub-index.</td>
</tr>
<tr>
<td>IXX_STORAGE</td>
<td>VARCHAR2(500)</td>
<td>Storage clause of the sub-index.</td>
</tr>
</tbody>
</table>

CTX_INDEX_SUB_LEXERS

This view shows the sub-lexers for each language for each index. It can be queried by CTXSYS.
### CTX_INDEX_SUB_LEXER_VALUES

Shows the sub-lexer attributes and their values. Accessible by CTXSYS.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISL_INDEX_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Index owner.</td>
</tr>
<tr>
<td>ISL_INDEX_NAME</td>
<td>VARCHAR2(30)</td>
<td>Index name.</td>
</tr>
<tr>
<td>ISL_LANGUAGE</td>
<td>VARCHAR2(30)</td>
<td>Language of sub-lexer</td>
</tr>
<tr>
<td>ISL_ALT_VALUE</td>
<td>VARCHAR2(30)</td>
<td>Alternate value of language</td>
</tr>
<tr>
<td>ISL_OBJECT</td>
<td>VARCHAR2(30)</td>
<td>Name of lexer object used for this language.</td>
</tr>
</tbody>
</table>

### CTX_INDEX_SUB_LEXER_VALUES

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISV_INDEX_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Index owner.</td>
</tr>
<tr>
<td>ISV_INDEX_NAME</td>
<td>VARCHAR2(30)</td>
<td>Index name.</td>
</tr>
<tr>
<td>ISV_LANGUAGE</td>
<td>VARCHAR2(30)</td>
<td>Language of sub-lexer</td>
</tr>
<tr>
<td>ISV_OBJECT</td>
<td>VARCHAR2(30)</td>
<td>Name of lexer object used for this language.</td>
</tr>
<tr>
<td>ISV_ATTRIBUTE</td>
<td>VARCHAR2(30)</td>
<td>Name of sub-lexer attribute</td>
</tr>
<tr>
<td>ISV_VALUE</td>
<td>VARCHAR2(500)</td>
<td>Value of attribute of sub-lexer.</td>
</tr>
</tbody>
</table>

### CTX_INDEX_VALUES

This view displays attribute values for each object used in indexes. This view is queryable by CTXSYS.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXV_INDEX_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Index owner.</td>
</tr>
<tr>
<td>IXV_INDEX_NAME</td>
<td>VARCHAR2(30)</td>
<td>Index name.</td>
</tr>
<tr>
<td>IXV_CLASS</td>
<td>VARCHAR2(30)</td>
<td>Class name.</td>
</tr>
<tr>
<td>IXV_OBJECT</td>
<td>VARCHAR2(30)</td>
<td>Object name.</td>
</tr>
<tr>
<td>IXV_ATTRIBUTE</td>
<td>VARCHAR2(30)</td>
<td>Attribute name</td>
</tr>
</tbody>
</table>
**CTX_OBJECTS**

This view displays all of the Text objects registered in the Text data dictionary. This view can be queried by any user.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJ_CLASS</td>
<td>VARCHAR2 (30)</td>
<td>Object class (Datastore, Filter, Lexer, and so on)</td>
</tr>
<tr>
<td>OBJ_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Object name</td>
</tr>
<tr>
<td>OBJ_DESCRIPTION</td>
<td>VARCHAR2 (80)</td>
<td>Object description</td>
</tr>
</tbody>
</table>

**CTX_OBJECT_ATTRIBUTES**

This view displays the attributes that can be assigned to preferences of each object. It can be queried by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAT_CLASS</td>
<td>VARCHAR2 (30)</td>
<td>Object class (Data Store, Filter, Lexer, and so on)</td>
</tr>
<tr>
<td>OAT_OBJECT</td>
<td>VARCHAR2 (30)</td>
<td>Object name</td>
</tr>
<tr>
<td>OAT_ATTRIBUTE</td>
<td>VARCHAR2 (64)</td>
<td>Attribute name</td>
</tr>
<tr>
<td>OAT_DESCRIPTION</td>
<td>VARCHAR2 (80)</td>
<td>Description of attribute</td>
</tr>
<tr>
<td>OAT_REQUIRED</td>
<td>VARCHAR2 (1)</td>
<td>Required attribute, either Y or N.</td>
</tr>
<tr>
<td>OAT_STATIC</td>
<td>VARCHAR2 (1)</td>
<td>Not currently used</td>
</tr>
<tr>
<td>OAT_DATATYPE</td>
<td>VARCHAR2 (64)</td>
<td>Attribute datatype. The value PROCEDURE indicates that the attribute of the object should be a stored procedure name.</td>
</tr>
<tr>
<td>OAT_DEFAULT</td>
<td>VARCHAR2 (500)</td>
<td>Default value for attribute</td>
</tr>
<tr>
<td>OAT_MIN</td>
<td>NUMBER</td>
<td>Minimum value</td>
</tr>
<tr>
<td>OAT_MAX</td>
<td>NUMBER</td>
<td>Maximum value</td>
</tr>
<tr>
<td>OAT_MAX_LENGTH</td>
<td>NUMBER</td>
<td>Maximum length</td>
</tr>
</tbody>
</table>
CTX_OBJECT_ATTRIBUTE_LOV

This view displays the allowed values for certain object attributes provided by Oracle Text. It can be queried by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAL_CLASS</td>
<td>NUMBER (38)</td>
<td>Class of object.</td>
</tr>
<tr>
<td>OAL_OBJECT</td>
<td>VARCHAR2 (30)</td>
<td>Object name.</td>
</tr>
<tr>
<td>OAL_ATTRIBUTE</td>
<td>VARCHAR2 (32)</td>
<td>Attribute name.</td>
</tr>
<tr>
<td>OAL_LABEL</td>
<td>VARCHAR2 (30)</td>
<td>Attribute value label.</td>
</tr>
<tr>
<td>OAL_VALUE</td>
<td>VARCHAR2 (64)</td>
<td>Attribute value.</td>
</tr>
<tr>
<td>OAL_DESCRIPTION</td>
<td>VARCHAR2 (80)</td>
<td>Attribute value description.</td>
</tr>
</tbody>
</table>

CTX_PARAMETERS

This view displays all system-defined parameters as defined by CTXSYS. It can be queried by any user.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| PAR_NAME   | VARCHAR2(30)      | Parameter name: max_index_memory
ctx_doc_key_type
default_index_memory
datastore
default_filter_binary
default_filter_text
default_filter_file
default_section_html
default_section_xml
default_section_text
default_lexer
default_stoplist
datastore
data_default
ctxcat_stoplist
ctxcat_storage
ctxcat_wordlist
ctxtcat_rule_lexer
ctxtcat_index_set
ctxtcat_rule_stoplist
ctxtcat_rule_storage
ctxtcat_rule_wordlist
log_directory
file_access_role |
| PAR_VALUE  | VARCHAR2(500)     | Parameter value. For max_index_memory and default_index_memory, PAR_VALUE stores a string consisting of the memory amount. For the other parameter names, PAR_VALUE stores the names of the preferences used as defaults for index creation. |
**CTX_PENDING**

This view displays a row for each of the user’s entries in the DML Queue. It can be queried by CTXSYS.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PND_INDEX_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>Index owner.</td>
</tr>
<tr>
<td>PND_INDEX_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Name of index.</td>
</tr>
<tr>
<td>PND_PARTITION_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Name of partition for local partition indexes.</td>
</tr>
<tr>
<td>PND_ROWID</td>
<td>ROWID</td>
<td>ROWID to be indexed.</td>
</tr>
<tr>
<td>PND_TIMESTAMP</td>
<td>DATE</td>
<td>Time of modification.</td>
</tr>
</tbody>
</table>

**CTX_PREFERENCES**

This view displays preferences created by Oracle Text users, as well as all the system-defined preferences included with Oracle Text. The view contains one row for each preference. It can be queried by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRE_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>Username of preference owner.</td>
</tr>
<tr>
<td>PRE_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Preference name.</td>
</tr>
<tr>
<td>PRE_CLASS</td>
<td>VARCHAR2 (30)</td>
<td>Preference class.</td>
</tr>
<tr>
<td>PRE_OBJECT</td>
<td>VARCHAR2 (30)</td>
<td>Object used.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRV_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>Username of preference owner.</td>
</tr>
<tr>
<td>PRV_PREFERENCE</td>
<td>VARCHAR2 (30)</td>
<td>Preference name.</td>
</tr>
<tr>
<td>PRV_ATTRIBUTE</td>
<td>VARCHAR2 (64)</td>
<td>Attribute name</td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Owner of the section group.</td>
</tr>
<tr>
<td>SEC_SECTION_GROUP</td>
<td>VARCHAR2(30)</td>
<td>Name of the section group.</td>
</tr>
<tr>
<td>SEC_TYPE</td>
<td>VARCHAR2(30)</td>
<td>Type of section, either ZONE, FIELD, SPECIAL, ATTR, STOP.</td>
</tr>
<tr>
<td>SEC_ID</td>
<td>NUMBER</td>
<td>Section id.</td>
</tr>
<tr>
<td>SEC_NAME</td>
<td>VARCHAR2(30)</td>
<td>Name of section.</td>
</tr>
<tr>
<td>SEC_TAG</td>
<td>VARCHAR2(64)</td>
<td>Section tag</td>
</tr>
<tr>
<td>SEC_VISIBLE</td>
<td>VARCHAR2(1)</td>
<td>Y or N visible indicator for field sections only.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGP_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Owner of section group.</td>
</tr>
<tr>
<td>SGP_NAME</td>
<td>VARCHAR2(30)</td>
<td>Name of section group.</td>
</tr>
<tr>
<td>SGP_TYPE</td>
<td>VARCHAR2(30)</td>
<td>Type of section group</td>
</tr>
</tbody>
</table>

**CTX_SQES**

This view displays the definitions for all SQEs that have been created by users. It can be queried by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRV_VALUE</td>
<td>VARCHAR2(500)</td>
<td>Attribute value</td>
</tr>
</tbody>
</table>
CTX_STOPLISTS

This view displays stoplists. Queryable by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQE_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>Owner of SQE.</td>
</tr>
<tr>
<td>SQE_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Name of SQE.</td>
</tr>
<tr>
<td>SQE_QUERY</td>
<td>VARCHAR2 (2000)</td>
<td>Query Text</td>
</tr>
</tbody>
</table>

CTX_STOPWORDS

This view displays the stopwords in each stoplist. Queryable by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPL_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>Owner of stoplist.</td>
</tr>
<tr>
<td>SPL_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Name of stoplist.</td>
</tr>
<tr>
<td>SPL_COUNT</td>
<td>NUMBER</td>
<td>Number of stopwords</td>
</tr>
<tr>
<td>SPL_TYPE</td>
<td>VARCHAR2 (30)</td>
<td>Type of stoplist, MULTI or BASIC.</td>
</tr>
<tr>
<td>SPW_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>Stoplist owner.</td>
</tr>
<tr>
<td>SPW_STOPLIST</td>
<td>VARCHAR2 (30)</td>
<td>Stoplist name.</td>
</tr>
<tr>
<td>SPW_TYPE</td>
<td>VARCHAR2 (10)</td>
<td>Stop type, either STOP_WORD, STOP_CLASS, STOP_THEME.</td>
</tr>
<tr>
<td>SPW_WORD</td>
<td>VARCHAR2 (80)</td>
<td>Stopword.</td>
</tr>
<tr>
<td>SPW_LANGUAGE</td>
<td>VARCHAR2 (30)</td>
<td>Stopword language.</td>
</tr>
</tbody>
</table>

CTX_SUB_LEXERS

This view contains information on multi-lexers and the sub-lexer preferences they contain. It can be queried by any user.
This view displays information about all the thesauri that have been created in the Text data dictionary. It can be queried by any user.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THS_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>Thesaurus owner.</td>
</tr>
<tr>
<td>THS_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Thesaurus name.</td>
</tr>
</tbody>
</table>

This view displays phrase information for all thesauri in the Text data dictionary. It can be queried by any user.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THP_THESAURUS</td>
<td>VARCHAR2 (30)</td>
<td>Thesaurus name.</td>
</tr>
<tr>
<td>THP_PHRASE</td>
<td>VARCHAR2 (256)</td>
<td>Thesaurus phrase.</td>
</tr>
<tr>
<td>THP_QUALIFIER</td>
<td>VARCHAR2 (256)</td>
<td>Thesaurus qualifier.</td>
</tr>
</tbody>
</table>

This view contains one row for each active trace, and shows the current value of each trace.
### 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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRC_ID</td>
<td>BINARY_INTEGER</td>
<td>Trace ID.</td>
</tr>
<tr>
<td>TRC_VALUE</td>
<td>NUMBER</td>
<td>Current trace value.</td>
</tr>
<tr>
<td>IDX_CHARSET_COLUMN</td>
<td>VARCHAR2(256)</td>
<td>Name of the charset column of base table.</td>
</tr>
<tr>
<td>IDX_DOCID_COUNT</td>
<td>NUMBER</td>
<td>Number of documents indexed.</td>
</tr>
<tr>
<td>IDX_FORMAT_COLUMN</td>
<td>VARCHAR2(256)</td>
<td>Name of the format column of base table.</td>
</tr>
<tr>
<td>IDX_ID</td>
<td>NUMBER</td>
<td>Internal index id.</td>
</tr>
<tr>
<td>IDX_KEY_NAME</td>
<td>VARCHAR(256)</td>
<td>Primary key column(s).</td>
</tr>
<tr>
<td>IDX_LANGUAGE_COLUMN</td>
<td>VARCHAR2(256)</td>
<td>Name of the language column of base table.</td>
</tr>
<tr>
<td>IDX_NAME</td>
<td>VARCHAR2(30)</td>
<td>Name of index.</td>
</tr>
<tr>
<td>IDX_STATUS</td>
<td>VARCHAR2(12)</td>
<td>Status, either INDEXED or INDEXING.</td>
</tr>
<tr>
<td>IDX_SYNC_INTERVAL</td>
<td>VARCHAR2(2000)</td>
<td>This is the interval string required by scheduler job. Only meaningful for AUTOMATIC sync. Always null for MANUAL and ON COMMIT sync.</td>
</tr>
<tr>
<td>IDX_SYNC_JOBNAME</td>
<td>VARCHAR2(50)</td>
<td>This is the scheduler job name for automatic sync. Only meaningful for AUTOMATIC sync and always null for other types of sync.</td>
</tr>
<tr>
<td>IDX_SYNC_MEMORY</td>
<td>VARCHAR2(100)</td>
<td>The sync memory size. Only meaningful for ON COMMIT and AUTOMATIC types of sync. For MANUAL sync, this is always null.</td>
</tr>
</tbody>
</table>
CTX_USER_INDEX_OBJECTS

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDX_SYNC_PARA_DEGREE</td>
<td>NUMBER</td>
<td>Degree of parallelism for sync. Only meaningful for the AUTOMATIC type of sync; always null for MANUAL and ON COMMIT syncs.</td>
</tr>
<tr>
<td>IDX_SYNC_TYPE</td>
<td>VARCHAR2(20)</td>
<td>Type of synching: AUTOMATIC, MANUAL or ON COMMIT.</td>
</tr>
<tr>
<td>IDX_TABLE</td>
<td>VARCHAR2(30)</td>
<td>Table name.</td>
</tr>
<tr>
<td>IDX_TABLE_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Owner of table.</td>
</tr>
<tr>
<td>IDX_TEXT_NAME</td>
<td>VARCHAR2(30)</td>
<td>Text column name.</td>
</tr>
<tr>
<td>IDX_TYPE</td>
<td>VARCHAR2(30)</td>
<td>Type of index: CONTEXT, CTXCAT, OR CTXRULE</td>
</tr>
</tbody>
</table>

CTX_USER_INDEX_ERRORS

This view displays the indexing errors for the current user and is queryable by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR_INDEX_NAME</td>
<td>VARCHAR2(30)</td>
<td>Name of index.</td>
</tr>
<tr>
<td>ERR_TIMESTAMP</td>
<td>DATE</td>
<td>Time of error.</td>
</tr>
<tr>
<td>ERR_TEXTKEY</td>
<td>VARCHAR2(18)</td>
<td>ROWID of errored document or name of errored operation (for example, ALTER INDEX)</td>
</tr>
<tr>
<td>ERR_TEXT</td>
<td>VARCHAR2(4000)</td>
<td>Error text.</td>
</tr>
</tbody>
</table>

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.
This view displays all index partitions for the current user. It is queryable by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXO_INDEX_NAME</td>
<td>VARCHAR2(30)</td>
<td>Name of index.</td>
</tr>
<tr>
<td>IXO_CLASS</td>
<td>VARCHAR2(30)</td>
<td>Object name</td>
</tr>
<tr>
<td>IXO_OBJECT</td>
<td>VARCHAR2(80)</td>
<td>Object description</td>
</tr>
<tr>
<td>IXP_DOCID_COUNT</td>
<td>NUMBER(38)</td>
<td>Number of documents associated with the index partition.</td>
</tr>
<tr>
<td>IXP_ID</td>
<td>NUMBER(38)</td>
<td>Index partition id.</td>
</tr>
<tr>
<td>IXP_INDEX_NAME</td>
<td>VARCHAR2(30)</td>
<td>Index name.</td>
</tr>
<tr>
<td>IXP_INDEX_PARTITION_NAME</td>
<td>VARCHAR2(30)</td>
<td>Index partition name.</td>
</tr>
<tr>
<td>IDX_SYNC_INTERVAL</td>
<td>VARCHAR2(2000)</td>
<td>This is the interval string required by scheduler job. Only meaningful for AUTOMATIC sync. Always null for MANUAL and ON COMMIT sync.</td>
</tr>
<tr>
<td>IDX_SYNC_JOBNAME</td>
<td>VARCHAR2(50)</td>
<td>This is the scheduler job name for automatic sync. It’s only meaningful for AUTOMATIC sync and always null for other types of sync.</td>
</tr>
<tr>
<td>IDX_SYNC_MEMORY</td>
<td>VARCHAR2(100)</td>
<td>The sync memory size. Only meaningful for ON COMMIT and AUTOMATIC types of sync. For MANUAL sync, this is always null.</td>
</tr>
<tr>
<td>IDX_SYNC_PARA_DEGREE</td>
<td>NUMBER</td>
<td>Degree of parallelism for sync. Only meaningful for the AUTOMATIC type of sync; always null for MANUAL and ON COMMIT syncs.</td>
</tr>
<tr>
<td>IDX_SYNC_TYPE</td>
<td>VARCHAR2(20)</td>
<td>Type of synching: AUTOMATIC, MANUAL or ON COMMIT.</td>
</tr>
</tbody>
</table>
This view displays all index set names that belong to the current user. It is queryable by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXS_NAME</td>
<td>VARCHAR2(30)</td>
<td>Index set name.</td>
</tr>
</tbody>
</table>

This view displays all the indexes in an index set that belong to the current user. It is queryable by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXX_INDEX_SET_NAME</td>
<td>VARCHAR2(30)</td>
<td>Index set name.</td>
</tr>
<tr>
<td>IXX_COLLIST</td>
<td>VARCHAR2(500)</td>
<td>Column list of the index.</td>
</tr>
<tr>
<td>IXX_STORAGE</td>
<td>VARCHAR2(500)</td>
<td>Storage clause of the index.</td>
</tr>
</tbody>
</table>

This view shows the sub-lexers for each language for each index for the querying user. This view can be queried by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISL_INDEX_NAME</td>
<td>VARCHAR2(30)</td>
<td>Index name.</td>
</tr>
<tr>
<td>ISL_LANGUAGE</td>
<td>VARCHAR2(30)</td>
<td>Language of sub-lexer</td>
</tr>
<tr>
<td>ISL_ALT_VALUE</td>
<td>VARCHAR2(30)</td>
<td>Alternate value of language.</td>
</tr>
</tbody>
</table>
**CTX_USER_INDEX_SUB_LEXER_VALS**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISL_OBJECT</td>
<td>VARCHAR2 (30)</td>
<td>Name of lexer object used for this language.</td>
</tr>
</tbody>
</table>

**CTX_USER_INDEX_SUB_LEXER_VALS**

Shows the sub-lexer attributes and their values for the querying user. This view can be queried by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISV_INDEX_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Index name.</td>
</tr>
<tr>
<td>ISV_LANGUAGE</td>
<td>VARCHAR2 (30)</td>
<td>Language of sub-lexer</td>
</tr>
<tr>
<td>ISV_OBJECT</td>
<td>VARCHAR2 (30)</td>
<td>Name of lexer object used for this language.</td>
</tr>
<tr>
<td>ISV_ATTRIBUTE</td>
<td>VARCHAR2 (30)</td>
<td>Name of sub-lexer attribute.</td>
</tr>
<tr>
<td>ISV_VALUE</td>
<td>VARCHAR2 (500)</td>
<td>Value of sub-lexer attribute</td>
</tr>
</tbody>
</table>

**CTX_USER_INDEX_VALUES**

This view displays attribute values for each object used in indexes for the current user. This view is queryable by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXV_INDEX_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Index name.</td>
</tr>
<tr>
<td>IXV_CLASS</td>
<td>VARCHAR2 (30)</td>
<td>Class name.</td>
</tr>
<tr>
<td>IXV_OBJECT</td>
<td>VARCHAR2 (30)</td>
<td>Object name.</td>
</tr>
<tr>
<td>IXV_ATTRIBUTE</td>
<td>VARCHAR2 (30)</td>
<td>Attribute name</td>
</tr>
<tr>
<td>IXV_VALUE</td>
<td>VARCHAR2 (500)</td>
<td>Attribute value.</td>
</tr>
</tbody>
</table>

**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.
This view displays all preferences defined by the current user. It can be queried by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRE_NAME</td>
<td>VARCHAR2(30)</td>
<td>Preference name.</td>
</tr>
<tr>
<td>PRE_CLASS</td>
<td>VARCHAR2(30)</td>
<td>Preference class.</td>
</tr>
<tr>
<td>PRE_OBJECT</td>
<td>VARCHAR2(30)</td>
<td>Object used.</td>
</tr>
</tbody>
</table>

This view displays all the values for preferences defined by the current user. It can be queried by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRV_PREFERENCE</td>
<td>VARCHAR2(30)</td>
<td>Preference name.</td>
</tr>
<tr>
<td>PRV_ATTRIBUTE</td>
<td>VARCHAR2(64)</td>
<td>Attribute name</td>
</tr>
<tr>
<td>PRV_VALUE</td>
<td>VARCHAR2(500)</td>
<td>Attribute value</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC__SECTION_GROUP</td>
<td>VARCHAR2(30)</td>
<td>Name of the section group.</td>
</tr>
</tbody>
</table>
###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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC_TYPE</td>
<td>VARCHAR2(30)</td>
<td>Type of section, either ZONE, FIELD, SPECIAL, STOP, or ATTR.</td>
</tr>
<tr>
<td>SEC_ID</td>
<td>NUMBER</td>
<td>Section id.</td>
</tr>
<tr>
<td>SEC_NAME</td>
<td>VARCHAR2(30)</td>
<td>Name of section.</td>
</tr>
<tr>
<td>SEC_TAG</td>
<td>VARCHAR2(64)</td>
<td>Section tag</td>
</tr>
<tr>
<td>SEC_VISIBLE</td>
<td>VARCHAR2(1)</td>
<td>Y or N visible indicator for field sections.</td>
</tr>
</tbody>
</table>

###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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQE_OWNER</td>
<td>VARCHAR2(30)</td>
<td>Owner of SQE.</td>
</tr>
<tr>
<td>SQE_NAME</td>
<td>VARCHAR2(30)</td>
<td>Name of SQE.</td>
</tr>
<tr>
<td>SQE_QUERY</td>
<td>VARCHAR2(2000)</td>
<td>Query Text</td>
</tr>
</tbody>
</table>

###CTX_USER_STOPLISTS

This view displays stoplists for current user. It is queryable by all users.
This view displays stopwords in each stoplist for current user. Queryable by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPL_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Name of stoplist.</td>
</tr>
<tr>
<td>SPL_COUNT</td>
<td>NUMBER</td>
<td>Number of stopwords</td>
</tr>
<tr>
<td>SPL_TYPE</td>
<td>VARCHAR2 (30)</td>
<td>Type of stoplist, MULTI or BASIC.</td>
</tr>
</tbody>
</table>

This view displays stopwords in each stoplist for current user. Queryable by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPW_STOPLIST</td>
<td>VARCHAR2 (30)</td>
<td>Stoplist name.</td>
</tr>
<tr>
<td>SPW_TYPE</td>
<td>VARCHAR2 (10)</td>
<td>Stop type, either STOP_WORD, STOP_CLASS, STOP_THEME.</td>
</tr>
<tr>
<td>SPW_WORD</td>
<td>VARCHAR2 (80)</td>
<td>Stopword.</td>
</tr>
<tr>
<td>SPW_LANGUAGE</td>
<td>VARCHAR2 (30)</td>
<td>Stopword language.</td>
</tr>
</tbody>
</table>

For the current user, this view contains information on multi-lexers and the sub-lexer preferences they contain. It can be queried by any user.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLX_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Name of the multi-lexer preference.</td>
</tr>
<tr>
<td>SLX_LANGUAGE</td>
<td>VARCHAR2 (30)</td>
<td>Language of the referenced lexer (full name, not abbreviation).</td>
</tr>
<tr>
<td>SLX_ALT_VALUE</td>
<td>VARCHAR2 (30)</td>
<td>An alternate value for the language.</td>
</tr>
<tr>
<td>SLX_SUB_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>Owner of the sub-lexer.</td>
</tr>
<tr>
<td>SLX_SUB_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Name of the sub-lexer.</td>
</tr>
</tbody>
</table>

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.
### CTX_USER_THESES_PHRASES

This view displays the phrase information of all thesaurus owned by the current user. It can be queried by all users.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THS_NAME</td>
<td>VARCHAR2 (30)</td>
<td>Thesaurus name</td>
</tr>
</tbody>
</table>

### CTX_VERSION

This view displays the CTXSYS data dictionary and code version number information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VER_DICT</td>
<td>CHAR (9)</td>
<td>The CTXSYS data dictionary version number.</td>
</tr>
<tr>
<td>VER_CODE</td>
<td>VARCHAR2 (9)</td>
<td>The version number of the code linked in to the Oracle Database shadow process. This column fetches the version number for linked-in code. Thus, you can use this column to detect and verify patch releases.</td>
</tr>
</tbody>
</table>
This appendix describes stopword transformations. The following topic is covered:

- Understanding Stopword Transformations

Understanding Stopword Transformations

When you use a stopword or stopword-only phrase as an operand for a query operator, Oracle Text rewrites the expression to eliminate the stopword or stopword-only phrase and then executes the query.

The following section describes the stopword rewrites or transformations for each operator. In all tables, the Stopword Expression column describes the query expression or component of a query expression, while the right-hand column describes the way Oracle Text rewrites the query.

The token **stopword** stands for a single stopword or a stopword-only phrase.

The token **non_stopword** stands for either a single non-stopword, a phrase of all non-stopwords, or a phrase of non-stopwords and stopwords.

The token **no_lex** stands for a single character or a string of characters that is neither a stopword nor a word that is indexed. For example, the + character by itself is an example of a no_lex token.

When the Stopword Expression column completely describes the query expression, a rewritten expression of **no_token** means that no hits are returned when you enter such a query.

When the Stopword Expression column describes a component of a query expression with more than one operator, a rewritten expression of **no_token** means that a no_token value is passed to the next step of the rewrite.
Transformations that contain a *no_token* as an operand in the *Stopword Expression* column describe intermediate transformations in which the *no_token* is a result of a previous transformation. These intermediate transformations apply when the original query expression has at least one stopword and more than one operator.

For example, consider the following compound query expression:

'(this NOT dog) AND cat'

Assuming that *this* is the only stopword in this expression, Oracle Text applies the following transformations in the following order:

1. stopword NOT non-stopword => no_token
2. no_token AND non_stopword => non_stopword

The resulting expression is:

'cat'

### Word Transformations

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>stopword</td>
<td>no_token</td>
</tr>
<tr>
<td>no_lex</td>
<td>no_token</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>non_stopword AND stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>non_stopword AND no_token</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>stopword AND non_stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>no_token AND non_stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
</tbody>
</table>
### OR Transformations

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stopword AND stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token AND stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>stopword AND no_token</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token AND no_token</code></td>
<td><code>no_token</code></td>
</tr>
</tbody>
</table>

### ACCUMulate Transformations

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>non_stopword ACCUM stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>non_stopword ACCUM no_token</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>stopword ACCUM non_stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>no_token ACCUM non_stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>stopword ACCUM stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token ACCUM stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>stopword ACCUM no_token</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token ACCUM no_token</code></td>
<td><code>no_token</code></td>
</tr>
</tbody>
</table>
### MINUS Transformations

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>non_stopword MINUS stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>non_stopword MINUS no_token</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>stopword MINUS non_stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token MINUS non_stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>stopword MINUS stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token MINUS stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>stopword MINUS no_token</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token MINUS no_token</code></td>
<td><code>no_token</code></td>
</tr>
</tbody>
</table>

### NOT Transformations

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>non_stopword NOT stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>non_stopword NOT no_token</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>stopword NOT non_stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token NOT non_stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>stopword NOT stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token NOT stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>stopword NOT no_token</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token NOT no_token</code></td>
<td><code>no_token</code></td>
</tr>
</tbody>
</table>

### EQUIVAlence Transformations

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>non_stopword EQUIV stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>non_stopword EQUIV no_token</code></td>
<td><code>non_stopword</code></td>
</tr>
</tbody>
</table>
### Stopword Transformations

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stopword EQUIV non_stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>no_token EQUIV non_stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>stopword EQUIV stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token EQUIV stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>stopword EQUIV no_token</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token EQUIV no_token</code></td>
<td><code>no_token</code></td>
</tr>
</tbody>
</table>

**Note:** When you use query explain plan, not all of the equivalence transformations are represented in the EXPLAIN table.

### NEAR Transformations

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>non_stopword NEAR stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>non_stopword NEAR no_token</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>stopword NEAR non_stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>no_token NEAR non_stopword</code></td>
<td><code>non_stopword</code></td>
</tr>
<tr>
<td><code>stopword NEAR stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token NEAR stopword</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>stopword NEAR no_token</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token NEAR no_token</code></td>
<td><code>no_token</code></td>
</tr>
</tbody>
</table>

### Weight Transformations

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stopword * n</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token * n</code></td>
<td><code>no_token</code></td>
</tr>
</tbody>
</table>
### Threshold Transformations

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stopword &gt; n</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token &gt; n</code></td>
<td><code>no_token</code></td>
</tr>
</tbody>
</table>

### WITHIN Transformations

<table>
<thead>
<tr>
<th>Stopword Expression</th>
<th>Rewritten Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stopword WITHIN section</code></td>
<td><code>no_token</code></td>
</tr>
<tr>
<td><code>no_token WITHIN section</code></td>
<td><code>no_token</code></td>
</tr>
</tbody>
</table>
Symbols

! operator, 3-42
- operator, 3-30
$ operator, 3-43
% wildcard, 3-57
* operator, 3-55
, operator, 3-9
= operator, 3-15
> operator, 3-48
? operator, 3-16
\ escape character, 4-2
_ wildcard, 3-57
| escape character, 4-2

A

ABOUT query, 3-5
example, 3-7
highlight markup, 8-16, 8-30
highlight offsets, 8-11, 8-28
viewing expansion, 10-6
accumulate operator, 3-9
scoring, 3-9
stopword transformations, H-3
ADD_ATTR_SECTION procedure, 7-3
ADD_EVENT procedure, 9-2
ADD_FIELD_SECTION procedure, 7-5
ADD_MDATA procedure, 7-11
ADD_MDATA_SECTION procedure, 7-14
ADD_SPECIAL_SECTION procedure, 7-16
ADD_STOP_SECTION procedure, 7-19
ADD_STOPCLASS procedure, 7-18
ADD_STOPTHEME procedure, 7-21
ADD_STOPWORD procedure, 7-22
ADD_SUB_LEXER procedure, 7-24
eexample, 2-46
ADD_TRACE procedure, 9-3
ADD_ZONE_SECTION procedure, 7-27
adding a trace, 9-3
adding an event, 9-2
adding metadata, 7-11, 7-14
AL32UTF8 character set, 2-48, 2-49, 2-50, 2-53
ALER TABLE
   UPDATE GLOBAL INDEXES, 1-19, 1-20, 1-21, 1-22
ALTER INDEX, 1-2
examples, 1-13
rebuild syntax, 1-3
rename syntax, 1-3
ALTER TABLE, 1-18
ALTER_PHRASE procedure, 12-3
ALTER_THESAURUS procedure, 12-5
alternate grammar template, 1-36
alternate language template, 1-36
alternate scoring template, 1-37
alternate spelling, 15-2
about, 15-1
base letter, 15-2
Danish, 15-5
disabling example, 7-64, 15-2
enabling example, 15-2
German, 15-5
normalized vs. original, 15-1
overriding, 15-4
Swedish, 15-5
alternate_spelling attribute, 2-45, 15-2
American
example, 2-81
BASIC_WORDLIST object
  attributes for, 2-72
  example, 2-77
BFILE column
  indexing, 1-44
BINARY
  format column value, 1-49
binary attribute, 2-8, 2-19
binary documents
  filtering, 2-5
BLOB column
  indexing, 1-44
  loading example, C-1
brace escape character, 4-2
brackets
  altering precedence, 3-4, 4-1
  grouping character, 4-1
broader term operators
  example, 3-12
broader term query feedback, 10-9
BROWSE_WORDS procedure, 10-2
browsing words in index, 10-2
BT function, 12-7
BT operator, 3-12
BTG function, 12-10
BTG operator, 3-12
BTI function, 12-12
BTI operator, 3-12
BTP function, 12-14
BTP operator, 3-12

C

case-sensitive
  ABOUT queries, 3-6
case-sensitive index
  creating, 2-42
CATSEARCH operator, 1-25
CHAR column
  indexing, 1-44
character sets
  Chinese, 2-48
  Japanese, 2-49
  Korean, 2-51, 2-53
characters
  continuation, 2-38
  numgroup, 2-39
  numjoin, 2-39
  printjoin, 2-39
  punctuation, 2-39
  skipjoin, 2-40
  specifying for newline, 2-41
  specifying for whitespace, 2-41
  startjoin and endjoin, 2-40
character-set
  indexing mixed columns, 2-22
  character-set conversion
    with INSO_FILTER, 2-26
  charset attribute, 2-22
  charset column, 1-48
CHARSET_FILTER
  attributes for, 2-21
    mixed character-set example, 2-23
Chinese
  fuzzy matching, 2-73
Chinese character sets supported, 2-48
Chinese lexicon, modifying, 14-11
Chinese text
  indexing, 2-48
CHINESE_VGRAM_LEXER object, 2-48
classifying documents, 6-2
  clustering, 2-86, 6-6
CLOB column
  indexing, 1-44
clump, 3-35
clump size in near operator, 3-34
  clustering, 2-86, 6-6
    hierarchical, 2-88
    KMEAN_CLUSTER, 2-87
types, 2-86
CLUSTERING procedure, 6-6
clustering types, 2-86
columns types
  supported for CTXCAT index, 1-58
  supported for CTXRULE index, 1-62
  supported for CTXXPATH index, 1-63
  supported for indexing, 1-44
command attribute, 2-30
compiler, lexical, 14-11
composite attribute
  BASIC_LEXER, 2-42
  KOREAN_MORP_LEXER, 2-54
composite textkey
  encoding, 8-22
composite word dictionary, 2-42
composite word index
  creating for German or Dutch text, 2-42
composite words
  viewing, 10-6
CONTAINS operator
  example, 1-38
  syntax, 1-34
CONTEXT index
  about, 1-43
  default parameters, 2-95
  syntax, 1-44
context index type, 1-43
continuation attribute, 2-38
control file example
  SQL*Loader, C-2
COUNT_HITS procedure, 10-5
CREATE_INDEX, 1-43
  CONTEXT, 1-44
  CTXCAT, 1-58
  CTXRULE, 1-61
  CTXXPATH, 1-63
  default parameters, 2-95
CREATE_INDEX_SCRIPT procedure, 11-5
CREATE_INDEX_SET procedure, 7-31, 7-65
CREATE_PHRASE procedure, 12-16
CREATE_POLICY procedure, 7-32
CREATE_PREFERENCE procedure, 7-35
CREATE_RELATION procedure, 12-18
CREATE_SECTION_GROUP procedure, 7-38
CREATE_STOPLIST procedure, 7-41
CREATE_THEASARUS function, 12-20
CREATE_TRANSLATION procedure, 12-21
creating an index report, 11-3
CTX_ADM package
  RECOVER, 5-2
  SET_PARAMETER, 5-3
CTX_CLASSES view, G-3
CTX_CLS
  CLUSTERING, 6-6
  TRAIN, 6-2
CTX_DDL package
  ADD_ATTR_SECTION, 7-3
  ADD_FIELD_SECTION, 7-5
  ADD_MDATA, 7-11
  ADD_MDATA_SECTION, 7-14
  ADD_SPECIAL_SECTION, 7-16
  ADD_STOP_SECTION, 7-19
  ADD_STOPCLASS, 7-18
  ADD_STOPTHEME, 7-21
  ADD_STOPWORD, 7-22
  ADD_SUB_LEXER, 7-24
  ADD_ZONE_SECTION, 7-27
  CREATE_INDEX_SET, 7-31, 7-65
  CREATE_POLICY, 7-32
  CREATE_PREFERENCE, 7-35
  CREATE_SECTION_GROUP, 7-38
  CREATE_STOPLIST, 7-41
  DROP_POLICY, 7-44
  DROP_PREFERENCE, 7-45
  DROP_STOPLIST, 7-47
  OPTIMIZE_INDEX procedure, 7-48
  REMOVE_MDATA, 7-53
  REMOVE_SECTION, 7-55
  REMOVE_STOPCLASS, 7-57
  REMOVE_STOPTHEME, 7-58
  REMOVE_STOPWORD, 7-59
  REPLACE_INDEX_METADATA, 7-60
  SET_ATTRIBUTE, 7-61
  SYNC_INDEX procedure, 7-62
  UNSET_ATTRIBUTE, 7-64
CTX_DOC package, 8-1
  FILTER, 8-3
  GIST, 8-6
  HIGHLIGHT, 8-11
  IFILTER, 8-14
  MARKUP, 8-16
  PKENCODE, 8-22
  POLICY_FILTER, 8-24
  POLICY_GIST, 8-25
  POLICY_HIGHLIGHT, 8-28
  POLICY_MARKUP, 8-30
  POLICY_THEMES, 8-33
  POLICY_TOKENS, 8-35
result tables, A-7
SET_KEY_TYPE, 8-37
THEMES, 8-38
TOKENS, 8-41
CTX_DOC_KEY_TYPE system parameter, 2-95
CTX_FEEDBACK_ITEM_TYPE type, A-6
CTX_FEEDBACK_TYPE type, 10-10, A-6
CTX_INDEX_ERRORS view, G-4
example, 1-58
CTX_INDEX_OBJECTS view, G-4
CTX_INDEX_SET_INDEXES view
views
CTX_INDEX_SET_INDEXES, G-5
CTX_INDEX_SUB_LEXERS view, G-5, G-18
CTX_INDEX_SUB_LEXERS_VALUES view, G-6
CTX_INDEX_VALUES view, G-6
CTX_INDEXES view, G-3
CTX_OBJECT_ATTRIBUTE_LOV view, G-8
CTX_OBJECT_ATTRIBUTES view, G-7
CTX_OBJECTS view, G-7
CTX_OUTPUT package, 9-1
ADD_EVENT, 9-2
ADD_TRACE, 9-3
END_LOG, 9-5
GET_TRACE_VALUE, 9-7
LOG_TRACES, 9-8
LOGFILENAME, 9-9
REMOVE_EVENT, 9-10
REMOVE_TRACE, 9-11
RESET_TRACE, 9-12
START_LOG, 9-13
CTX_PARAMETERS view, 2-94, G-8
CTX_PENDING view, G-10
CTX_PREFERENCE_VALUES view, G-10
CTX_PREFERENCES view, G-10
CTX_QUERY package
BROWSE_WORDS, 10-2
COUNT_HITS, 10-5
EXPLAIN, 10-6
HFEEDBACK, 10-9
REMOVE_SQE, 10-14
result tables, A-1
STORE_SQE, 10-15
CTX_QUERY.disable_transactional_query session
variable, 1-53
CTX_REPORT output format, 11-3, 11-4, 11-8,
11-10, 11-20
CTX_REPORT package, 11-1
CREATE_INDEX_SCRIPT, 11-5
CREATE_POLICY_SCRIPT, 11-6
DESCRIBE_INDEX, 11-3
DESCRIBE_POLICY, 11-4
function versions of procedures, 11-2
INDEX_SIZE, 11-7
INDEX_STATS, 11-9
QUERY_LOG_SUMMARY, 11-14
TOKEN_INFO, 11-19
TOKEN_TYPE, 11-21
CTX_SECTION_GROUPS view, G-11
CTX_SECTIONS view, G-11
CTX_SQES view, G-11
CTX_STOPLISTS view, G-12
CTX_STOPWORDS view, G-12
CTX_SUB_LEXERS view, G-12
CTX_THES package, 12-1
ALTER_PHRASE, 12-3
ALTER_THESAURUS, 12-5
BT, 12-7
BTG, 12-10
BTI, 12-12
BTP, 12-14
CREATE_PHRASE, 12-16
CREATE_RELATION, 12-18
CREATE_THESAURUS, 12-20
DROP_PHRASE, 12-22
DROP_RELATION, 12-24
DROP_THESAURUS, 12-27
NT, 12-30
NTG, 12-33
NTI, 12-35
NTP, 12-37
OUTPUT_STYLE, 12-39
PT, 12-40
result tables, A-10
RT, 12-42
SN, 12-44
SYN, 12-45
THES_TT, 12-48
TR, 12-49
TRSYN, 12-51
TT, 12-53
CTX_THESAURIS view, G-13
CTX_THES.CREATE_TRANSLATION, 12-21
CTX_THES.DROP_TRANSLATION, 12-28
CTX_THES.UPDATE_TRANSLATION, 12-55
CTX_TRACE_VALUES view, G-13
CTX_ULEXER package, 13-1
CTX_USER_INDEX_ERRORS view, G-15
example, 1-58
CTX_USER_INDEX_OBJECTS view, G-15
CTX_USER_INDEX_SET_INDEXES view, G-17
CTX_USER_INDEX_SETS view, G-17
CTX_USER_INDEX_SUB_LEXERS view, G-17
CTX_USER_INDEX_VALUES view, G-18
CTX_USER_INDEXES view, G-14
CTX_USER_PENDING view, G-18
CTX_USER_PREFERENCE_VALUES view, G-19
CTX_USER_PREFERENCES view, G-19
CTX_USER_SECTION_GROUPS view, G-20
CTX_USER_SECTIONS view, G-19
CTX_USER_SQES view, G-20
CTX_USER_STOPLISTS view, G-20
CTX_USER_STOPWORDS view, G-21
CTX_USER_SUB_LEXERS view, G-21
CTX_USER_THESES_PHRASES view, G-22
CTX_USER_THESAURIS view, G-21
CTX_VERSION view, G-22
CTXCAT index
about, 1-43
default parameters, 2-96
supported preferences, 1-59
syntax, 1-58
unsupported preferences, 1-60
txkbtc compiler, 14-5
txlc (lexical compiler), 14-11
txload, 14-1
examples, 14-4
import file structure, C-4
CTXRULE index
about, 1-43
default parameters, 2-97
syntax, 1-61
CTXXPATH index
about, 1-44
syntax, 1-63

CTXXPATH indextype
creating, 1-64

D
Danish
alternate spelling, 15-5
index defaults, 2-92
supplied stoplist, E-3
data storage
defined procedurally, 2-15
direct, 2-3
eample, 7-35
external, 2-10
master/detail, 2-8
URL, 2-12
datastore types, 2-3
DATE column, 1-44
DBMS_PCLUTIL
BUILD_PART_INDEX, 1-57
default index
eample, 1-53
default parameters
changing, 2-98
CONTEXT index, 2-95
CTXCAT index, 2-96
CTXRULE index, 2-97
viewing, 2-97
DEFAULT thesaurus, 3-13, 3-32
DEFAULT_CTXCAT_INDEX_SET system
parameter, 2-96
DEFAULT_CTXCAT_LEXER system
parameter, 2-97
DEFAULT_CTXCAT_STOPLIST system
parameter, 2-97
DEFAULT_CTXCAT_STORAGE system
parameter, 2-96
DEFAULT_CTXCAT_WORDLIST system
parameter, 2-96
DEFAULT_CTXRULE_LEXER system
parameter, 2-97
DEFAULT_CTXRULE_STOPLIST system
parameter, 2-97
DEFAULT_CTXRULE_WORDLIST system
parameter, 2-97
composite word indexing, 2-42
fuzzy matching, 2-73
index defaults, 2-92
stemming, 2-72
supplied stoplist, E-3

email
  filtering and indexing, 2-27
empty indexes
  creating, 1-51
EMPTY_STOPLIST system-defined
  preference, 2-93
enabling tracing, 9-3
END_LOG procedure, 9-5
END_QUERY_LOG procedure, 9-6
ending a log, 9-5
ending a query log, 9-6
endjoins attribute, 2-40
English
  fuzzy matching, 2-73
  index defaults, 2-92
  supplied stoplist, E-1
english attribute (Korean lexer), 2-54
environment variables
  setting for Inso filter, B-3
equivalence operator, 3-15
  stopword transformations, H-4
  with NEAR, 3-35
errors
  indexing, 1-58
  escaping special characters, 4-2
event
  adding, 9-2
  removing, 9-10
EVERY parameter, 1-7, 1-51
example, 1-56
EXP_TAB table type, A-10
expansion operator
  soundex, 3-42
  stem, 3-43
  viewing, 10-6
EXPLAIN procedure, 10-6
  example, 10-8
result table, A-1
explain table
  creating, 10-7
  retrieving data example, 10-8
structure, A-1
extending knowledge base, 14-5
external filters
  specifying, 2-30

failed index operation
  resuming, 1-9
fast filtering, 2-25, 2-28
field section
  defining, 7-5
  limitations, 7-7
  querying, 3-59
field sections
  adding dynamically, 1-11
  repeated, 3-62
  WITHIN example, 3-61
file data storage
  example, 7-35
FILE_DATASTORE object, 2-10
example, 2-11
FILE_DATASTORE system-defined
  preference, 2-91
filter attribute
  MULTI_COLUMN_DATASTORE, 2-5
filter formats
  supported, B-3
FILTER procedure, 8-3
  example, 8-5
  in-memory example, 8-4
  result table, A-7
filter table
  structure, A-7
filter types, 2-21
filtering
  fast, with INSO_OUTPUT_FORMATTING
    attribute, 2-28
  fast, with OUTPUT_FORMATTING
    attribute, 2-25
  multi_column_datastore, 2-5
stored procedures, 2-31
  to plain text, 8-14
  to plain text and HTML, 8-3
filters
  character-set, 2-21
  Inso, 2-23, B-1
  user, 2-30
Finnish
  index defaults, 2-92
  supplied stoplist, E-4
  format column, 1-49
  formatted documents
    filtering, 2-23
  fragmentation of index, 1-51
French
  fuzzy matching, 2-73
  supplied stoplist, E-5
  French stemming, 2-72
  ftp_proxy attribute, 2-14
fuzzy matching
  automatic language detection, 2-73
  example for enabling, 2-77
  specifying a language, 2-74
fuzzy operator, 3-16
fuzzy_match attribute, 2-74
fuzzy_numresults attribute, 2-75
fuzzy_score attribute, 2-74

G

German
  alternate spelling attribute, 2-45
  alternate spelling conventions, 15-5
  composite word indexing, 2-42
  fuzzy matching, 2-73
  index defaults, 2-92
  new spelling, querying with, 2-45, 15-3
  stemming, 2-72
  supplied stoplist, E-6
GET_TRACE_VALUE procedure, 9-7
Gist
  generating, 8-25
gist
  generating, 8-6
GIST procedure
  example, 8-9
  result table, A-7
  updated syntax, 8-6
Gist table
  structure, A-7

H

hanja attribute, 2-54
HASPATH operator, 3-18
HFEEDBACK procedure, 10-9
  example, 10-10
  result table, A-4
hierarchical clustering, 2-88
hierarchical query feedback information
  generating, 10-9
hierarchical relationships
  in thesaurus import file, C-6
HIERARCHY_DEPTH attribute, 2-88
HIGHLIGHT procedure, 8-11
  example, 8-13
  result table, A-8
highlight table
  example, 8-13
  structure, A-8
highlighting
  generating markup, 8-16, 8-30
  generating offsets, 8-11, 8-28
  with NEAR operator, 3-37
hit counting, 10-5
HOME environment variable
  setting for INSO, B-3
homographs
  in broader term queries, 3-13
  in queries, 3-12
  in thesaurus import file, C-7
HTML
  bypassing filtering, 2-25
  filtering to, 8-3
  generating, 8-24
  generating highlight offsets for, 8-11, 8-28
  highlight markup, 8-16, 8-30
  highlighting example, 8-20
  indexing, 1-62, 2-27, 2-81, 7-38
  zone section example, 7-28
index creation, 1-43
creating a report on, 11-3
creating index script, 11-5
describing, 11-3
duplicating with script, 11-5
renaming, 1-3
script, 11-5
show size of objects, 11-7
show statistics, 11-9
synchronizing, 1-7, 1-51
transactional, 10-5
transactional CONTEXT, 1-8, 1-52
viewing registered, G-3
index creation
    custom preference example, 1-54
default example, 1-53
index creation parameters
element, 2-81
index errors
deleting, 1-58
viewing, 1-58
index fragmentation, 1-51
index maintenance, 1-2
index objects, 2-1
viewing, G-4, G-7
index optimization, 1-9
index preference
    about, 2-1
    creating, 2-2, 7-35
index reports, 11-1
index requests
    logging, 9-13
index tokens
    generating for a document, 8-35, 8-41
INDEX_PROCEDURE user_lexer attribute, 2-57
INDEX_SIZE procedure, 11-7
INDEX_STATS procedure, 11-9
index_stems attribute, 2-44
index_text attribute, 2-44
index_themes attribute, 2-43
indexing
    master/detail example, 2-10
    multilingual documents, 2-46, 2-71, D-5
    parallel, 1-4, 1-46
    themes, 2-43
indexing types
    classifier, 2-84
    clustering, 2-86
datastore, 2-3
filter, 2-21
lexer, 2-36
section group, 2-81
storage, 2-79
vs. preferences, 2-2
wordlist, 2-72
indexless document services, see policy-based
document services
indextype context, 1-43
inflectional stemming
    enabling, 2-74
INPATH operator, 3-20
INPUT_TYPE user_lexer attribute, 2-58
INSERT statement
    loading example, C-1
Inso filter
    index preference object, 2-23
    setting up, B-1
    supported formats, B-3
supported platforms, B-2
unsupported formats, B-11
INSO_FILTER object, 2-23
and transactional CONTEXT indexes, 1-53
character-set conversion, 2-26
INSO_FILTER system-defined preference, 2-91
INSO_OUTPUT_FORMATTING attribute, 2-28
inverse frequency scoring, F-1
Italian
fuzzy matching, 2-73
stemming, 2-72
supplied stoplist, E-7

J
JA16EUC character set, 2-49, 2-50
JA16EUCTILDE character set, 2-49, 2-50
JA16EUCYEN character set, 2-49, 2-50
JA16S JIS character set, 2-49, 2-50
JA16S JISTILDE character set, 2-49, 2-50
JA16S JISYEN character set, 2-49, 2-50
Japanese
fuzzy matching, 2-73
index defaults, 2-92
indexing, 2-49
stemming, 2-72
japanese attribute (Korean lexer), 2-54
Japanese character sets supported, 2-49
Japanese EUC character set, 2-49
Japanese lexicon, modifying, 14-11
Japanese stemming, 2-72, 3-43
JAPAN ESE LEXER, 2-49
JAPANESE_YGRAM LEXER object, 2-49
JOB_QUEUE_PROCESSES initialization
parameter, 1-46

K
k_table_clause attribute, 2-80
KMEAN_CLUSTER object, 2-87
knowledge base
supported character set, 14-5
user-defined, 14-9
knowledge base extension compiler, 14-5
KO16KSC5601 character set, 2-51, 2-53
KO16MSWIN949 character set, 2-53
Korean
fuzzy matching, 2-73
index defaults, 2-92
unicode character support, 2-53
korean character sets supported, 2-51, 2-53
Korean text
indexing, 2-52
KOREAN LEXER object, 2-51
KOREAN_MORP LEXER, 2-52
composite example, 2-55
supplied dictionaries, 2-52
KOREAN_MORPH LEXER
Unicode support, 2-53

L
language
setting, 2-36
language column, 1-50
left-truncated searching
improving performance, 2-75
lexer types, 2-36
lexical compiler, 14-11
lexicon. See entries under dictionary
loading text
SQL INSERT example, C-1
SQL*Loader example, C-1
loading thesaurus, 14-1
LOB columns
loading, C-1
local partition index
parallelism, 1-56
local partitioned index, 1-45
LOG_DIRECTORY system parameter, 2-94, 9-9
LOG_TRACES procedure, 9-8
LOGFILENAME procedure, 9-9
logging
ending, 9-5
ending a log, 9-6
getting log file name, 9-9
index requests, 9-13
logging queries, 11-14
logging traces, 9-8
logical operators
with NEAR, 3-35
LONG columns
  indexing, 1-44
long_word attribute, 2-54

M
mail filter configuration file, 2-29
mail filtering, see email, 2-27
MAIL_FILTER object, 2-27
MAIL_FILTER_CONFIG_FILE system parameter, 2-29
maintaining index, 1-2
MARKUP procedure, 8-16
  example, 8-20
    HTML highlight example, 8-20
  result table, A-8
markup table
  example, 8-20
  structure, A-8
master/detail data storage, 2-8
  example, 2-8, 7-36
master/detail tables
  indexing example, 2-10
MATCHES operator, 1-66
MAX_INDEX_MEMORY system parameter, 2-94
max_span parameter in near operator, 3-34
maxthreads attribute, 2-13
maxurls attribute, 2-14
MDATA operator, 3-27
MDATA section, 7-11, 7-14, 7-53
memory
  for index synchronize, 1-10
  for indexing, 1-10, 1-50, 1-63, 7-62
META tag
  creating field sections for, 7-7
  creating zone section for, 7-28
metadata, 1-6, 3-27
  replacing, 7-60
METADATA keyword, 1-6
ALTER INDEX example, 1-15
metadata section, 7-11, 7-14, 7-53
MINUS operator, 3-30
  stopword transformations, H-4
mixed character-set columns
indexing, 2-22
mixed_case attribute, 2-42
mixed-format columns
  filtering, 2-23
  indexing, 2-25
  supported formats for, B-3
modifying user dictionary, 14-11
morpheme attribute, 2-54
MULTI_LEXER object
  CREATE INDEX example, 1-55
  example, 2-46
MULTI_LEXER type, 2-46
MULTI_STOPLIST type, 7-41
multi-language indexing, 2-46, 2-71, 7-24, D-5
multi-language stoplist, 2-46, 2-89
multi-language tables
  querying, 1-41, 2-47
multi-lexer example
  migrating from single language, 1-15

N
n_table_clause attribute, 2-80
narrower term operators
  example, 3-31
narrower term query feedback, 10-9
NEAR operator
  backward compatibility, 3-36
    highlighting, 3-37
    scoring, 3-35
  stopword transformations, H-5
    with other operators, 3-35
    with within, 3-60
nested section searching, 3-61
nested zone sections, 7-30
nested_column attribute, 2-18
NESTED_DATASTORE attribute, 2-19
NESTED_DATASTORE object, 2-18
nested_lineno attribute, 2-19
nested_text attribute, 2-19
nested_type attribute, 2-19
new_german_spelling attribute, 2-45, 15-3
newline attribute, 2-41
NEWS_SECTION_GROUP object, 2-82, 7-39
no_proxy attribute, 2-14
p_table_clause, 2-80
PARAGRAPH keyword, 3-62
paragraph section
  defining, 7-16
  querying, 3-59
parallel index creation, 1-56
parallel indexing, 1-4, 1-46
  DBMS_PCLUTIL.BUILD_PART_INDEX, 1-57
    example, 1-56
    local partitioned index, 1-46
parameter
  transactional, 1-8, 1-52
parameters
  setting, 5-3
  viewing system-defined, G-8
parentheses
  altering precedence, 3-4, 4-1
  grouping character, 4-1
partitioned index
  creating local in parallel, 1-46
    example, 1-55
    local, 1-45
    parallel creation, 1-57
    rebuild example, 1-13
partitioned index creation
  example, 1-56
partitioned tables
  modifying, 1-18
path attribute, 2-11
PATH environment variable
  setting for Inso, B-3
PATH_SECTION_GROUP
  querying with, 3-20
PATH_SECTION_GROUP object, 2-82, 7-39
PATH_SECTION_GROUP system-defined
  preference, 2-93
pending DML
  viewing, G-10
performance
  wildcard searches, 3-58
PKENCODE function, 8-22
plain text
  bypassing filtering, 2-25
filtering to, 8-3, 8-14
highlight markup, 8-16, 8-30
indexing with NULL_FILTER, 2-27
offsets for highlighting, 8-11
policy, 8-1
  create script, 11-6
  duplicate with script, 11-6
  report describing, 11-4
POLICY_FILTER procedure, 8-24
POLICY_GIST procedure, 8-25
POLICY_HIGHLIGHT procedure, 8-28
POLICY_MARKUP procedure, 8-30
POLICY_THEMES procedure
  syntax, 8-33
POLICY_TOKENS procedure
  syntax, 8-35
policy-based document services, 8-1
populate index parameter, 1-51
populate parameter, 1-51
Portuguese
  supplied stoplist, E-8
precedence of operators, 3-2
  altering, 3-4, 4-1
  equivalence operator, 3-15
    example, 3-3
    viewing, 10-6
preference classes
  viewing, G-3
preference values
  viewing, G-10
preferences
  about, 2-1
  changing, 1-6
  creating, 7-35
  dropping, 7-45
  replacing, 1-3
  specifying for indexing, 1-47
  system-defined, 2-90
  viewing, G-10
  vs. types, 2-2
preferred term operator
  example, 3-40
prefix_index attribute, 2-75
prefix_length_max attribute, 2-76
prefix_length_min attribute, 2-76
printjoins attribute, 2-39
privileges
  required for indexing, 1-44
procedure
  CTX_DDL.REPLACE_INDEX_METADATA, 7-60
  CTX_OUTPUT_LOG_TRACES, 9-8
  CTX_OUTPUT.ADD_TRACE, 9-3
  CTX_OUTPUT.END_QUERY_LOG, 9-6
  CTX_OUTPUT.GET_TRACE_VALUE, 9-7
  CTX_OUTPUT.REMOVE_TRACE, 9-11
  CTX_OUTPUT.RESET_TRACE, 9-12
procedure attribute, 2-16
PROCEDURE_FILTER object, 2-31
progressive relaxation template, 1-35
prove_themes attribute, 2-43
proximity operator, see NEAR operator
PT function, 12-40
PT operator, 3-40
punctuations attribute, 2-39

Q
query
  accumulate, 3-9
  analysis, 11-14
  AND, 3-11
  broader term, 3-12
  equivalence, 3-15
  example, 1-38
  hierarchical feedback, 10-9
  MINUS, 3-30
  narrower term, 3-31
  NOT, 3-38
  on unsynched index, 1-52
  OR, 3-39
  preferred term, 3-40
  related term, 3-41
  report of logged, 11-14
  stored, 3-45
  synonym, 3-46
  threshold, 3-48
  top term, 3-53
  transactional, 1-52, 10-5
  translation term, 3-49
  translation term synonym, 3-51
  weighted, 3-55
query relaxation template, 1-35
query rewrite template, 1-34
query template, 1-27, 1-34
QUERY_LOG_SUMMARY procedure, 11-14
QUERY_PROCEDURE user_lexer attribute, 2-60
R
r_table_clause attribute, 2-80
rebuilding index
  example, 1-13
  syntax, 1-3
RECOVER procedure, 5-2
related term operator, 3-41
related term query feedback, 10-9
relaxing queries, 1-35
relevance ranking
  word queries, F-1
REMOVE_EVENT procedure, 9-10
REMOVE_MDATA procedure, 7-53
REMOVE_SECTION procedure, 7-55
REPLACE_INDEX_METADATA procedure, 7-60
replacing, 1-6
replacing metadata, 1-6
replacing preferences, 1-3
report
  describing index, 11-3
  describing policy, 11-4
  index objects, 11-7
  index size, 11-7
  index statistics, 11-9
  of logged queries, 11-14
  token information, 11-19

Index-15
reserved words and characters, 4-3
escaping, 4-2
RESET_TRACE procedure, 9-12
resetting a trace, 9-12
result table
  TOKENS, A-9
result tables, A-1
  CTX_DOC, A-7
  CTX_QUERY, A-1
  CTX_THES, A-10
resuming failed index, 1-9
  example, 1-13
rewriting queries, 1-34
RFC 1738 URL specification, 2-12
RFC-2045 messages
  filtering, 2-27
RFC-822 messages
  filtering, 2-27
RT function, 12-42
RT operator, 3-41
RULE_CLASSIFIER type, 2-84
rules
  generating, 6-2

S
Salton's formula for scoring, F-1
scope notes
  finding, 12-44
SCORE operator, 1-69
scoring
  accumulate, 3-9
  effect of DML, F-3
  for NEAR operator, 3-35
scoring algorithm
  word queries, F-1
script
  create index, 11-5
  create policy, 11-6
section group
  creating, 7-38
  viewing information about, G-11
section group example, 2-82
section group types, 2-81, 7-38
section searching, 3-59

  nested, 3-61
sections
  adding dynamically, 1-3
  constraints for dynamic addition, 1-13
  creating attribute, 7-3
  creating field, 7-5
  creating zone, 7-27
  nested, 7-30
  overlapping, 7-29
  removing, 7-55
  repeated field, 7-7
  repeated zone, 7-29
  viewing information on, G-11
SENTENCE keyword, 3-62
sentence section
  defining, 7-16
  querying, 3-59
SET_ATTRIBUTE procedure, 7-61
SET_KEY_TYPE procedure, 8-37
SET_PARAMETER procedure, 2-94, 5-3
show size of index objects, 11-7
Simplified Chinese
  index defaults, 2-92
single-byte languages
  indexing, 2-36
  skipjoins attribute, 2-40
SN procedure, 12-44
soundex operator, 3-42
Spanish
  fuzzy matching, 2-73
  stemming, 2-72
  supplied stoplist, E-8
special section
  defining, 7-16
  querying, 3-59
spelling
  alternate, 15-2
  base letter, 15-2
  new German, 15-3
  overriding alternate, 15-4
  spelling, alternate, 15-1
  spelling, new German, 2-45
SQE operator, 3-45
SQL commands
  ALTER INDEX, 1-2
CREATE INDEX, 1-43
DROP INDEX, 1-65
SQL operators
  CONTAINS, 1-34
  SCORE, 1-69
SQL*Loader
  example, C-1
  example control file, C-2
  example data file, C-3
sqlldr example, C-2
START_LOG procedure, 9-13
startjoins attribute, 2-40
statistics, showing index, 11-9
stem indexing, 2-44
stem operator, 3-43
stemmer attribute, 2-74
stemming, 2-72, 2-74, 3-43
  automatic, 2-72
  example for enabling, 2-77
stop section
  adding dynamically, 1-12
  dynamically adding example, 1-16
stop sections
  adding, 7-19
  stop_dic attribute, 2-54
stopclass
  defining, 7-18
  removing, 7-57
stoplist
  creating, 7-41
  Danish, E-3
  dropping, 7-47
  Dutch, E-3
  English, E-1
  Finnish, E-4
  French, E-5
  German, E-6
  Italian, E-7
  modifying, 2-90
  multi-language, 2-46, 2-89
  Portuguese, E-8
  Spanish, E-8
  Swedish, E-9
stoplists
  about, 2-89
  creating, 2-89
  viewing, G-12
stoptheme
  defining, 7-21
  removing, 7-58
stopword
  adding dynamically, 1-3, 1-11
  defining, 7-22
  removing, 7-59
  viewing all in stoplist, G-12
stopword transformation, H-1
  viewing, 10-6
stopwords
  adding dynamically, 2-90
  removing, 2-90
storage defaults, 2-80
storage index preference
  example, 7-36
storage objects, 2-79
STORE_SQE procedure
  example, 3-45
  syntax, 10-15
stored queries, 3-45
stored query expression
  creating, 10-15
  removing, 10-14
  viewing, G-20
  viewing definition, G-11
sub-lexer values
  viewing, G-6
sub-lexers
  viewing, G-5, G-12, G-17, G-18
substring index
  example for creating, 2-77
substring_index attribute, 2-75
supplied stoplists, E-1
Swedish
  alternate spelling, 15-5
  index defaults, 2-92
  supplied stoplist, E-9
SYN function, 12-45
SYN operator, 3-46
SYNC EVERY parameter, 1-7, 1-51
SYNC ON COMMIT parameter, 1-7, 1-51
sync parameter, 1-7, 1-51
SYNC_INDEX procedure, 7-62
synchronize index, 1-7, 1-51
synonym operator, 3-46
system parameters, 2-94
defaults for indexing, 2-95
system recovery
    manual, 5-2
system-defined preferences, 2-90

T

table structure
    explain, A-1
    filter, A-7
    Gist, A-7
    hfeedback, A-4
    highlight, A-8
    markup, A-8
    theme, A-9
tagged text
    searching, 3-59
template query, 1-27, 1-34
TEXT
    format column value, 1-49
text column
    supported types, 1-44
Text data dictionary
    cleaning up, 5-2
text-only index
    enabling, 2-44
    example, 7-35
theme functionality
    supported languages, 14-9
theme highlighting
    generating markup, 8-16
    generating offsets, 8-11, 8-28
    HTML markup example, 8-21
    HTML offset example, 8-13
theme index
    as default in English, 2-92
    creating, 2-43
theme proving
    enabling, 2-43
theme summary
    generating, 8-6
generating top n, 8-9
theme table
    structure, A-9
theme_language attribute, 2-44
themes
    generating for document, 8-33, 8-38
    generating highlight markup, 8-16, 8-30
    highlight offset example, 8-13
    indexing, 2-43
    obtaining top n, 8-40
THEMES procedure
    result table, A-9
    syntax, 8-38
THES_TT procedure, 12-48
thesaurus
    compiling, 14-5
    creating, 12-20
    creating relationships, 12-16
    DEFAULT, 3-13
    dropping, 12-27
    import/export examples, 14-4
    importing/exporting, 14-1
    procedures for browsing, 12-1
    renaming and truncating, 12-5
    viewing information about, G-13
thesaurus import file
    examples, C-9
    structure, C-4
thesaurus phrases
    altering, 12-3
    dropping, 12-22
thesaurus relations
    creating, 12-18
    dropping, 12-24
thesaurus scope note
    finding, 12-44
thesaurus top terms
    finding all, 12-48
threshold operator, 3-48
    stopword transformations, H-6
timeout attribute, 2-13
to_upper attribute, 2-54
token index optimization, 1-9
token report, generating, 11-19
token, translating name into, 11-21
TOKEN_INFO procedure, 11-19
TOKEN_TYPE procedure, 11-21
TOKENS procedure
  result table, A-9
  syntax, 8-41
top term operator, 3-53
TR function, 12-49
TR operator, 3-49
trace value
  getting, 9-7
traces, available, 9-3
tracing
  adding a trace, 9-3
  available traces, 9-3
  CTX_TRACE_VALUES view, G-13
  enabling, 9-3
  getting trace values, 9-7, G-13
  logging traces, 9-8
  removing trace, 9-11
  resetting trace, 9-12
TRAIN procedure, 6-2
transactional CONTEXT index, 1-8, 1-52
transactional index, 10-5
transactional parameter, 1-8, 1-52
transactional text query, 1-8, 1-52
  disabling, 1-53
transformation
  stopword, H-1
translation term operator, 3-49
translation term synonym operator, 3-51
translations
  adding to thesaurus, 12-21
  dropping, 12-28
  English name to numeric token, 11-21
  updating, 12-55
TRSYN function, 12-51
TRSYN operator, 3-51
TT function, 12-53
TT operator, 3-53
type
  MULTI_LEXER, 2-46
  WORLD_LEXER, 2-71, D-5
types, 2-2
  indexing, 2-2
  see also indexing types

U
unicode support in Korean lexer, 2-53
UNIX platforms
  setting variables for Inso, B-3
UNSET_ATTRIBUTE procedure, 7-64
unsupervised classification, see clustering
UPDATE GLOBAL INDEXES, 1-19, 1-20, 1-21, 1-22
UPDATETranslation procedure, 12-55
URL syntax, 2-12
URL_DATASTORE object
  attributes for, 2-12
  example, 2-15
URL_DATASTORE system-defined
  preference, 2-91
urlsize attribute, 2-13
user dictionary, modifying, 14-11
USER_DATASTORE object, 2-15
  example, 2-17
USER_DATSTORE
  filtering binary documents, 8-14
  user_dic attribute, 2-54
USER_FILTER object, 2-30
  example, 2-31
USER_LEXER object, 2-56
UTF-16 endian auto-detection, 2-22
UTF8, 2-49
UTF8 character set, 2-37, 2-48, 2-49, 2-50, 2-51, 2-53
utilities
  ctxload, 14-1

V
VARCHAR2 column
  indexing, 1-44
verb_adjective attribute, 2-54
version numbers
  viewing, G-22
viewing
  operator expansion, 10-6
  operator precedence, 10-6
views, G-1
  CTX_CLASSES, G-3
  CTX_INDEX_ERRORS, G-4
  CTX_INDEX_OBJECTS, G-4

Index-19
adding dynamically, 1-11
creating, 7-27
dynamically adding example, 1-16
querying, 3-59
repeating, 7-29