

UNLOCKING HIDDEN GEMS IN ORACLE TEXT

Bill Coulam, Church of Jesus Christ of Latter Day Saints

With a finite amount of time and brain power, we have to pick and choose the technologies we try, reject and adopt. Personally I've focused on PL/SQL, tuning and modeling, but purposely avoided large subproducts from Oracle, like Spatial, Workflow, interMedia, Warehouse Builder, and Oracle Text, thinking they were all irrelevant to my employers and clients.

Ignoring Oracle Text was a big mistake. Although it is a large and complex product, meant for efficiently searching and cataloging massive libraries of textual content, it sports a number of features that are accessible and useful to anyone with data to query and expose.

At one point or another, in one role or another, Oracle professionals will be involved in projects that require the ability to search and report on enterprise data. Frequently the data customer would eagerly adopt advanced search capabilities if they only knew the option existed, capabilities like case and diacritic insensitivity, proximity, similar spelling, visibility inside attachments like PDF and Word, etc. Some of these features can be built in the middle tier with open source and commercial libraries. Some can be written, at great expense, by hand. But all of these features are already installed and licensed for use in your Oracle database (SE, SE One, PE and EE). Why re-invent the wheel? It only takes a few minutes to start using them! Awareness of Oracle Text features should make you a more valuable modeler, analyst, developer, manager or DBA.

This paper will attempt to guide those new to Oracle Text through installation, creation and use of basic CONTEXT indexes¹, customization for advanced capabilities, and architecture of multi-column and multi-table indexes. These features should be useful to any shop, including those that are OLTP-centric with no document collections. CTXCAT, CTXRULE and CTXXPATH indexes will not be addressed here, nor will we cover CATSEARCH or MATCHES queries.

Checking Installation

Databases created with DBCA will have Oracle Text already installed by default. If tests aren't working or the database was created manually, look for an account named CTXSYS. At one time Oracle Text was named Oracle ConText, and its basic indexes are still called context indexes. So the cryptic account name means Context System administration account. CTXSYS should contain about 260 objects in 9iR2 and about 340 objects in 10gR2. All the objects should be VALID. The account will probably be locked. Unlock it if you wish to explore CTXSYS objects and metadata. If the password is not known, and the database has yet to be hardened, try "ctxsys" for the default password. In production there is no need to connect as CTXSYS to use Text (10g and up); the account can remain locked.

If examples still aren't working see if the database contains an old version of Text that wasn't upgraded properly. Use the following to see which version of Oracle Text is installed:

```
SELECT * FROM ctxsys.ctx_version;
```

Installing Oracle Text

If the CTXSYS account is not complete or empty, drop the account first. Then as SYS run catctx.sql, found in ORACLE_HOME, under ctx/admin. This script takes four arguments: *password*, *default_tablespace*, *temp_tablespace*, and LOCK|NOLOCK.

```
conn SYS/syspasswd AS SYSDBA
@?/ctx/admin/catctx.sql mypasswd SYSAUX TEMP NOLOCK
```

Now connect as the CTXSYS user and run the default language preferences script, drdefxx.sql, found in ctx/admin/defaults, where *xx* is one of English(us), Danish(dk), Dutch(nl), Finnish(sf), French(f), German(d), Italian(it), Portuguese(pt), Spanish(e), and Swedish(s). For most of us in the United States, we would run

```
conn CTXSYS/mypasswd
@?/ctx/admin/defaults/drdefus.sql
```

¹ For the grammar pedantic, yes, *indices* is the correct plural of index.

Context Indexes

Assume an account named EDW (Enterprise Data Warehouse), and a reference table named PLACES, (see the Appendix for DDL and sample data). Using default settings, the simplest Text index to create² would be on a single column:

```
CREATE INDEX place_nm_cidx ON places(place_nm) INDEXTYPE IS CTXSYS.CONTEXT; COMMIT;
```

That's it! That's all it takes. The awesome power of CONTAINS queries and operators are now at your fingertips.

Underneath the covers, Oracle Text initiated the datastore portion of its engine to fetch and retrieve the text to be processed. The text was then fed through a filter which understands document types and formatting. The sectioner was next in line in case the text had been organized by means of tags or sections (like XML). Then the text was passed to the lexer which understands language nuances and breaks up the text into tokens. Finally, the indexer takes over and creates the underlying tables and metadata which govern the index, its parameters, keys and content.

Tour of a Context Index

A context index is not a single Oracle object, but rather 4-6 tables starting with DR\$*indexname* in the index-owning account. The names of these tables cannot be changed. Observe the new contents of the EDW account. We see a \$I table which holds our column's tokens, and indicates how many times that token is found in the collection, and the range of documents where the token is found. The \$K table will have one row per "document", mapping the context document to the original ROWID for speedy table access. The \$N table is the negative list used to track deleted documents. The \$R table is known as the rowid table. If enabling speedy substring searches (a feature of the wordlist preference), a \$P IOT table will be created as well. New with 11g, a \$S table will be created if using the FILTER BY or ORDER BY parameters, which we won't discuss here.

The \$K table and token_first/token_last columns of the \$I table use DOCID, a numeric value given to each unique indexed "document." To map the original rows to the DOCID, join rowid to the keymap table:

```
SELECT k.docid, p.* FROM dr$place_nm_cidx$k k, places p WHERE k.textkey = p.rowid;
```

Context index creation also leaves metadata in CTXSYS tables. The place_nm_cidx created above used default settings and dropped new data in dr\$index, dr\$index_value and dr\$index_object, which are more accessible through the ctx* views:

```
-- index metadata
SELECT * FROM ctx_indexes WHERE idx_name = 'PLACE_NM_CIDX';
-- parameters used and their values (default values in this case)
SELECT * FROM ctx_index_objects t WHERE ixo_index_name = 'PLACE_NM_CIDX';
-- attributes used
SELECT * FROM ctx_index_values t WHERE ixv_index_name = 'PLACE_NM_CIDX';
```

CONTAINS Queries

A context index enables CONTAINS queries. CONTAINS is like INSTR, except it is many times more powerful. CONTAINS takes two or three parameters: *indexed column*, *query_expression* and optionally an integer *label*. CONTAINS returns a numeric relevance score between 0 and 100 for every row selected. You will never see 0. If Text determines the row's relevance is 0, the row is not included in the result set. The SCORE operator returns the relevance as a virtual column. SCORE just requires a numeric label which matches the CONTAINS label. To get results with most relevant matches first, order by SCORE(*label*) DESC. The *query_expression* fed to CONTAINS can be part of the CONTEXT or CTXCAT grammar. We will look solely at the operators in CONTEXT grammar.

Rather than repeat what the Oracle Text Reference says, let's look at a number of CONTAINS operators in action on our sample data. The shameless Monty Python references were added to render some of the operators meaningful on the simple

² On earlier versions of Oracle the indexing account may need the CTXAPP role (created when Oracle Text was installed) to create Text indexes. This is not true in 10g and above.

place_nm column. Although the examples are somewhat trivial, the power of these search features becomes more apparent when using them on real document collections and large character columns. Real-world, large volume document collections are difficult, if not impossible, to mine and search effectively without Oracle Text.

One more index, this one on the large HTML pages in the CLOB column, is required for the examples to function.

```
CREATE INDEX place_notes_cidx ON places(place_notes) INDEXTYPE IS CTXSYS.CONTEXT; COMMIT;

-- Simple CONTAINS finds records where the term is included at least once.
SELECT place_nm FROM places t WHERE CONTAINS(place_nm,'Egg') > 0;
-- SCORE with optional, matching CONTAINS label, returns the relevance as a column
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'Egg',1) > 0;
-- "Mixed" or Structured CONTAINS (regular SQL predicate with CONTAINS operator)
SELECT place_nm, place_type_nm FROM places t WHERE CONTAINS(place_nm,'Toledo') > 0 AND place_type_nm
= 'City';
-- ABOUT looks for themes in an indexed document
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_notes,'about(monarchy)',1) > 0;
-- ACCUM scores doc better by number of times ACCUM term(s) appear(s), a term can be weighted
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'spam',1) > 0;
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'bacon*3 ACCUM spam',1) > 0;
-- alternative ACCUM syntax is term1,term2
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'bacon*3,spam',1) > 0;
-- CONTAINS queries can use Boolean operators, parenthesis and order of operations
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'Egg & Spam | Bacon',1) > 0;
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'(Egg AND Spam) OR Bacon',1) > 0;
-- EQUIV helps with alternate spellings or names the user might be unaware of
-- Scania is a transliteration of Sweden's Skåne
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'Skåne=Scania',1) > 0;
-- FUZZY(?) helps with misspellings and alternate spellings, finding similar word forms
-- Syntax: fuzzy(term, [score], [numresults], [weight|noweight]) Old Syntax: ?term
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_notes,'?united',1) > 0;
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_notes,'fuzzy(united,40,5,W)',1) > 0;
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_notes,'fuzzy(united)',1) > 0;
-- MINUS(-) excludes documents that contain the unwanted term
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'spam - egg',1) > 0;
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'spam MINUS egg',1) > 0;
-- NOT(~), similar to MINUS searches for one term, but excludes the other
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'spam ~ egg',1) > 0;
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'spam NOT (egg OR bacon)',1) > 0;
-- NEAR(;) enables word proximity searching
-- Syntax: NEAR((term1,term2,...,termN)[,max_span[,order]]) Old Syntax: word1 ; word2
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'islands;northern',1) > 0;
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'NEAR((islands,northern),4)',1) > 0;
-- to ensure matches with words in the order specified, third parameter must be TRUE
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'NEAR((northern,islands),4,TRUE)',1)
> 0;
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_notes,'NEAR((northern,islands),4,TRUE)
AND commonwealth',1) > 0 ORDER BY SCORE(1) DESC;
-- SOUNDEX(!) operator expands to words that have similar sounds (works best in English)
-- search term misspelled by user, but found Spain anyway
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'!Spain',1) > 0;
-- STEM($) expands to related terms with the same linguistic root
-- BASIC_LEXER (the default) supports English, French, Spanish, Italian, German, and Dutch
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_notes,'$arab',1) > 0;
-- THRESHOLD(>) brings back matches whose expression or word score is greater than threshold
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_nm,'Toledo*8 > 30',1) > 0;
-- WILDCARD(%_) does left %term, right term%, and double-truncated wildcard searches %term%
SELECT place_nm, SCORE(1) FROM places t WHERE CONTAINS(place_notes,'%wealth%',1) > 0;
-- Multiple CONTAINS in a single query
SELECT place_nm, place_type_nm FROM places t WHERE CONTAINS(place_nm,'Toledo',1) > 0 AND
CONTAINS(place_notes,'Moorish',2) > 0;
-- If search terms include any of the above keywords or symbols, they will need to be escaped with {}
for strings and \ for individual characters.
```

Some operators, like SOUNDEX and wildcard, resemble capabilities in existing SQL grammar. The real power comes when fluidly combining these operators, like a query expression that uses proximity, boolean, SOUNDEX and stemming at the

same time. That would be impossible in traditional SQL, and nearly impossible and certainly too slow in PL/SQL. There is so much more to CONTEXT grammar, like thesaurus-based searches, stored query expressions, special XML operators, and translation terms. Sadly space does not permit. Oracle Text is vast. This paper only scratches the surface, but hopefully exposes the layer you will find immediately useful and accessible.

The examples above all function based on default Oracle Text preferences. Even more power is exposed when Text preferences are customized to meet enterprise needs. In fact, Oracle recommends you do not rely on the default settings except for testing purposes.

Customizing Your Index

Ordinarily context indexes are meant for CLOB, BLOB, BFILE, XMLType or URIType columns to index textual and binary documents, ordinarily inaccessible to SQL. In some shops, these documents number in the millions. Indexing them requires careful storage, content and refresh design. Luckily, Oracle Text is highly flexible. There are many options, and if one of the built-ins isn't enough, constructing custom preferences is fairly easy.

We will take a look at the options that could be useful to shops not already expert with Oracle Text. These options are specified by the PARAMETERS clause of the CREATE INDEX statement.

Syntax of the context index PARAMETERS clause:

```
CREATE INDEX... name ON table(column(s))
  INDEXTYPE IS CTXSYS.CONTEXT PARAMETERS ('
[DATASTORE datastore_pref]
[FILTER filter_pref]
[CHARSET COLUMN charset_column_name]
[FORMAT COLUMN format_column_name]
[LEXER lexer_pref]
[LANGUAGE COLUMN language_column_name]
[WORDLIST wordlist_pref]
[STORAGE storage_pref]
[STOPLIST stoplist]
[SECTION GROUP section_group]
[MEMORY memsize]
[POPULATE | NOPOPULATE] -- 11g
[SYNC (MANUAL | EVERY "interval" | ON COMMIT)]
[TRANSACTIONAL]')
```

For the parameters that take a preference, stoplist or section group, one first calls the appropriate create routine in CTX_DDL, setting attributes. Then reference the named parameter with an option in the PARAMETERS clause. See the examples starting on page 6.

Datstores

Use datastore types to tell Oracle where and how the text is stored. The default DIRECT_DATASTORE is used for text stored in a single column. FILE_DATASTORE is for text stored on a file system accessible to Oracle Text. URL_DATASTORE is for tables that store addresses to documents stored on the intra/internet. If the text you want to index as a single virtual document is stored over multiple columns in the same table, use MULTI_COLUMN_DATASTORE. If stored over multiple rows, as in a master-detail table relationship, use DETAIL_DATASTORE. USER_DATASTORE relies on a user-created PL/SQL routine to synthesize text from various sources when indexing.

Filters

The default filters of Oracle Text (AUTO_FILTER for binary columns, NULL_FILTER for character columns) are sufficient for most document collections. A filter gives Text the know-how to process HTML, XML, plain text, formatted documents (RTF, PDF, etc.) and word processor documents, separating the characters from the tags and binary bytes. If the documents are in a character set foreign to the database, or in a mixed character set, use the CHARSET_FILTER. If the documents are copies of RFC-822 compliant email messages, use the MAIL_FILTER. There are other filters that are less useful for most.

Lexers

A lexer understands languages and their nuances. Lexers do most of the heavy lifting, breaking text up into tokens right before indexing. The default BASIC_LEXER is sufficient for English, French, German and other western languages. The AUTO_LEXER may be more appropriate for documents written in one of the other 32 supported major languages. If the collection of documents is written in multiple languages, the MULTI_LEXER or WORLD_LEXER should help. There are also special lexers for Chinese, Japanese and Korean. Finally the USER_LEXER lets you implement your own custom lexer.

Some of the attributes of the BASIC and AUTO_LEXER are particularly useful. If original word case needs to be preserved, use the mixed_case attribute. If diacritics are a problem, use the base_letter attribute. The printjoins attribute can preserve hyphenated words and words with underscores as whole tokens. There are several other attributes to help the lexer do a better job of finding the start and end of words and sentences, but I've found the default settings to suffice 98% of the time.

Format Column

This parameter is optionally used in conjunction with the filters to explicitly tell Oracle Text whether the document in the current row is BINARY, TEXT or should be IGNOREd. This is useful if the document collection contains a mix of notes, word processor docs, HTML pages, screenshots, jpg pictures, etc. Text can auto-detect the format, but the performance of indexing can be improved by setting all the binary, non-text documents to IGNORE so they aren't even inspected.

Sectioners

A section group tells Oracle how to tag indexed text. The default NULL_SECTION_GROUP only sections by sentence and paragraph and does not use tags. The HTML_SECTION_GROUP knows how to interpret and tag HTML documents. The AUTO, PATH and XML section groups are all used for XML documents, XML_SECTION_GROUP enables WITHIN queries. PATH_SECTION_GROUP enables the INPATH and HASPATH operators. Finally there is the NEWS_SECTION_GROUP meant for RFC 1036 compliant newsgroup posts.

Stoplist

Stoplists identify words that are to be skipped and not indexed, common words like an, the, it, etc. If the default stoplist for a language is not sufficient, a custom one can be created using CTX_DDL.create_stoplist.

Wordlists

Wordlist attributes allow one to exert a higher degree of control over how text is indexed to improve the operation and performance of features like stemming, fuzzy matching, wildcard expansion, and prefix and/or substring indexing. The latter two attributes add significantly to the size and maintenance overhead of the context index parts. But if users only have, or are only willing to use, the first few letters of words, like doc%, or only portions of words, like %olog% to find matches, creating a wordlist preference with prefix_index set to YES or substring_index set to TRUE is a good idea. The length of the generated prefixes can be controlled³.

Storage

Since some document collections are enormous, it is important to control the storage attributes of the DR\$ tables behind the index, including the LOB storage for the BLOB column in the \$R table. A storage preference is created, named and given attributes. The most useful attributes are the i_table_clause, i_index_clause, k_table_clause, r_table_clause. Oracle recommends the i_index_clause retain the default COMPRESS 2 setting for optimal performance. They also recommend the r_table_clause retain the default "LOB(DATA) STORE AS (CACHE)". There are other storage attributes less useful for the masses. However, if you decide to index token substrings to speed up partial word searches, the p_table_clause will come in handy for the DR\$*indexname*\$P IOT table created for the substrings.

Keeping a Text Index Fresh

Oracle Text indexes are best on fairly static data. The moment text in the source table is updated, inserted or deleted, the index is stale and can't give 100% accurate results. The index can be kept in sync with changes, but at a cost of index fragmentation. Eventually the index must be rebuilt. Like materialized views, these limitations must be understood by all, especially if Text is employed in transactional databases. There are numerous options available to the architect to optimize the resync and rebuild, including parallel, partitioning, DBMS_PCLXUTIL and more. We will only look at three basic items.

At any time in 11g and below, the simplest way to bring an index up to speed with recent changes is to call sync_index().

³ There is a bug in the Oracle docs from 11g and down. The attributes that control prefix length are prefix_min_length and prefix_max_length, not prefix_length_min and prefix_length_max as specified.

```
exec ctx_ddl.sync_index('place_nm_cidx');
```

Oracle provides a script, named drjobdm.sql, to make sync_index call automation easier. As the index owner, one would call it passing *indexname* and *interval minutes*:

```
@?/ctx/sample/script/drjobdml.sql place_nm_cidx 60
```

To correct eventual index fragmentation, the index can be rebuilt from scratch or optimized in fast or full mode. FAST mode compacts fragmented rows but does not remove old data. FULL optimizes with old data removed:

```
exec ctx_ddl.optimize_index('place_nm_cidx', 'FULL');
```

New with 10g is the ability to keep the index in sync with changes transparently. This can be put on an interval using the parameter SYNC EVERY “*interval*”, or if DML activity is low, consider SYNC ON COMMIT. If DML activity is high though, SYNC ON COMMIT would render the text index horribly fragmented. Using SYNC EVERY “*interval*” accomplishes the same thing as the pre-10g sync job. Under the covers, a job is automatically submitted for the index owner, so the owner needs CREATE JOB granted to it for either sync mechanism to function.

Also new with 10g is the TRANSACTIONAL parameter, which allows unsynchronized changes to be immediately included in a context index. This parameter tells Oracle to search the existing text index, then do a dynamic, in-memory index of the unsynchronized data to ensure the latest data isn’t left behind. A resync is still eventually required, as dynamically indexing unsynchronized data will drastically slow performance over time.

Multi-Column Index

It is time to put these definitions to good use. Some applications have useful textual data spread across multiple columns in a table. For example, the user might want a single “Name Search” field, but the source table has seven different fields for title, first, middle, last, maiden, legal, and preferred names. Another scenario is the requirement for a single address search, but the source data is a wide address table with 3-4 columns for each type of address: home, business, and billing. To implement such searches, developers typically create views, materialized views, or columns on the source table where the original data is concatenated into a single field, or less common, run one query on each column for the user-supplied term.

A cleaner approach however, is to use Oracle Text to create a multi-column index. Let’s try this with a simple table that has customer first, middle and last names. Let’s assume that some names are hyphenated, so we want to preserve them as single tokens instead of two or more. We want our index to be diacritic-insensitive, since not all users know how to enter accents, umlauts, cedillas, etc. Finally, some users will only know part of the name’s correct spelling, usually the first part, so we want to speed up right-truncated searches with a prefixed index as well. We just create a few preferences, then the index:

```
EXEC ctx_ddl.drop_preference('cust_multids');
EXEC ctx_ddl.drop_preference('cust_lexer');
EXEC ctx_ddl.drop_preference('cust_wordlist');
EXEC ctx_ddl.drop_preference('cust_storage');
BEGIN
  ctx_ddl.create_preference('cust_multids', 'MULTI_COLUMN_DATASTORE');
  ctx_ddl.set_attribute('cust_multids', 'columns', 'first_nm|CHR(32)||mid_nm|CHR(32)||
last_nm|CHR(32)||email_addr as fullstring');

  ctx_ddl.create_preference('cust_lexer', 'BASIC_LEXER');
  ctx_ddl.set_attribute('cust_lexer', 'printjoins', '- '); -- keeps hyphenation
  ctx_ddl.set_attribute('cust_lexer', 'base_letter', 'YES'); -- removes diacritics

  ctx_ddl.create_preference('cust_wordlist', 'BASIC_WORDLIST');
  ctx_ddl.set_attribute('cust_wordlist', 'prefix_index', 'TRUE');
  ctx_ddl.set_attribute('cust_wordlist', 'prefix_min_length', '3');
  ctx_ddl.set_attribute('cust_wordlist', 'prefix_max_length', '6');

  ctx_ddl.create_preference('cust_storage', 'BASIC_STORAGE');
  ctx_ddl.set_attribute('cust_storage', 'i_table_clause', 'TABLESPACE &&cust_data');
  ctx_ddl.set_attribute('cust_storage', 'k_table_clause', 'TABLESPACE &&cust_data');
  ctx_ddl.set_attribute('cust_storage', 'n_table_clause', 'TABLESPACE &&cust_data');
```

```

    ctx_ddl.set_attribute('cust_storage','r_table_clause','TABLESPACE &&cust_data LOB (data) STORE AS
(cache)');
    ctx_ddl.set_attribute('cust_storage','i_index_clause','TABLESPACE &&cust_index COMPRESS 2');
END;
/

DROP INDEX cust_strings_cidx;
DROP TABLE customer CASCADE CONSTRAINTS;
CREATE TABLE customer(cust_id NUMBER PRIMARY KEY, first_nm VARCHAR2(100),
    mid_nm VARCHAR2(100), last_nm VARCHAR2(100), email_addr VARCHAR2(100));
INSERT INTO customer VALUES(1, 'John','Morgan','Smith','smithjm@harpers.com');
INSERT INTO customer VALUES(2, 'Morgan',NULL,'Smythe','morgan_smythe@reuters.com');
INSERT INTO customer VALUES(3, 'William','Anjo','Morgan','william-a-morgan@hotmail.com');
INSERT INTO customer VALUES(4, 'Rip','William','Van-winkle','rip.vanwinkle@narcoleptics.org');
COMMIT;

-- Since Text needs to track updates to a single column, we have to give it one
ALTER TABLE customer ADD otx_upd_flg VARCHAR2(1 BYTE) DEFAULT 'N';
-- And update it whenever the text changes
CREATE OR REPLACE TRIGGER customer_biu
    BEFORE UPDATE OR INSERT ON customer
    FOR EACH ROW
DECLARE
BEGIN
    IF (INSERTING OR UPDATING) THEN
        :new.otx_upd_flg := 'Y';
    END IF;
END customer_biu;
/

CREATE INDEX cust_strings_cidx ON customer(otx_upd_flg)
    INDEXTYPE IS CTXSYS.CONTEXT
    PARAMETERS ('DATASTORE cust_multids
        LEXER cust_lexer
        WORDLIST cust_wordlist
        STORAGE cust_storage');
COMMIT; -- necessary after Text index creation

```

With the index in place, we can try some CONTAINS queries, referencing the dummy column as if that were the indexed column:

```

-- user can't remember spelling
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'!smith') > 0;
-- user only has "Morgan" to go on to find customer
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'morgan') > 0;
-- user wants to find hyphenated names (hyphens and underscores need to be escaped)
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'van\-winkle') > 0;
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'morgan\_smythe') > 0;
-- user only has part of the customer's email address
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'Narc%') > 0;
-- user has name portions, but not sure about order
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'NEAR((Smythe,Morgan))') > 0;
-- user has name portions and is sure about their order
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'NEAR((Morgan,Smith),1,TRUE)') > 0;
-- user doesn't know how to enter the nordic diacritical marks
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'Anjo') > 0;

```

Multi-Column, Multi-Table Index

Occasionally development will get a requirement to provide a single-field search screen (not unlike Google) which searches all the text fields within a given data model or submodel. This is another area where Oracle Text can save the day, this time with a user-defined datastore, where development provides a custom routine to concatenate and section the text from its various sources into a single virtual document for use by the filter, sectioner, lexer and indexer. Let's reuse the customer table, and add some customer contact notes in a child table to demonstrate this feature in its simplest form possible.

```

DROP INDEX cust_strings_cidx FORCE;

DROP TABLE customer_contact;

```

```

CREATE TABLE customer_contact (cust_id NUMBER NOT NULL REFERENCES customer(cust_id),
  contact_dt DATE NOT NULL, contact_type VARCHAR2(10 BYTE) CHECK (contact_type IN (
    'EMAIL','INCALL','OUTCALL','LETTER')), note VARCHAR2(4000));
INSERT INTO customer_contact VALUES (1,SYSDATE-5,'INCALL','Called to check warranty length.');
```

```

INSERT INTO customer_contact VALUES (2,SYSDATE-4,'EMAIL','Wants callback at home, 435-234-5555');
INSERT INTO customer_contact VALUES (2,SYSDATE-3,'OUTCALL','Called. Spoke to Veronica Smythe.');
```

```

INSERT INTO customer_contact VALUES (3,SYSDATE-2,'LETTER','Sent latest privacy notice.');
```

```

INSERT INTO customer_contact VALUES (4,SYSDATE-1,'LETTER','Sent 81st late payment notice.');
```

```

INSERT INTO customer_contact VALUES (4,SYSDATE,'INCALL','Fell asleep for 20 yrs. Wants interest
waived.');
```

```

COMMIT;

CREATE OR REPLACE PACKAGE cust_util AS
PROCEDURE concat_columns(i_rowid IN ROWID, io_text IN OUT NOCOPY VARCHAR2);
END cust_util;
/
CREATE OR REPLACE PACKAGE BODY cust_util AS
PROCEDURE concat_columns(i_rowid IN ROWID, io_text IN OUT NOCOPY VARCHAR2)
AS
  lr_cust customer%ROWTYPE;

  CURSOR cur_cust (i_cust_id IN customer.cust_id%TYPE) IS
  SELECT first_nm, mid_nm, last_nm, email_addr FROM customer
  WHERE cust_id = i_cust_id;

  CURSOR cur_notes (i_cust_id IN customer_contact.cust_id%TYPE) IS
  SELECT note FROM customer_contact WHERE cust_id = i_cust_id;

  PROCEDURE add_piece(i_add_str IN VARCHAR2) IS
    lx_too_big EXCEPTION;
    PRAGMA EXCEPTION_INIT(lx_too_big, -6502);
  BEGIN
    io_text := io_text||' '||i_add_str;
    EXCEPTION WHEN lx_too_big THEN NULL; -- silently don't add the string.
  END add_piece;

BEGIN
  BEGIN
    SELECT * INTO lr_cust FROM customer WHERE ROWID = i_rowid;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN;
  END;

  add_piece('<FULLNAME>'||lr_cust.first_nm||CHR(32)||
    lr_cust.mid_nm||CHR(32)||
    lr_cust.last_nm||'</FULLNAME>');
  add_piece('<EMAIL_ADDR>'||lr_cust.email_addr||'</EMAIL_ADDR>');

  -- Now collect text from any calls or letters from the customer
  FOR lr_note IN cur_notes(lr_cust.cust_id) LOOP
    add_piece('<NOTE>'||lr_note.note||'</NOTE>');
  END LOOP;

END concat_columns;
END cust_util;
/

EXEC ctx_ddl.drop_section_group('cust_sectioner');
EXEC ctx_ddl.drop_preference('cust_user_ds');
BEGIN
  ctx_ddl.create_section_group('cust_sectioner', 'BASIC_SECTION_GROUP');
  ctx_ddl.add_field_section('cust_sectioner', 'fullname', 'fullname', TRUE);
  ctx_ddl.add_field_section('cust_sectioner', 'email_addr', 'email_addr', TRUE);
  ctx_ddl.add_field_section('cust_sectioner', 'note', 'note', TRUE);

  ctx_ddl.create_preference('cust_user_ds', 'USER_DATASTORE');
  ctx_ddl.set_attribute('cust_user_ds', 'procedure',
sys_context('userenv','current_schema')||'.'||'cust_util.concat_columns');
  ctx_ddl.set_attribute('cust_user_ds', 'output_type', 'VARCHAR2');
```

```

END;
/

-- Again add a trigger or two to update otx_upd_flg when columns in the index change
-- <snipped to save space, just copy and modify from previous example>

CREATE INDEX cust_strings_cidx ON customer(otx_upd_flg)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('DATASTORE cust_user_ds
              SECTION GROUP cust_sectioner
              LEXER cust_lexer
              STORAGE cust_storage
              SYNC (EVERY "SYSDATE+6/24")
              TRANSACTIONAL');

COMMIT;

```

We can now efficiently search all the columns at once, as if they were all part of one column, no matter if the search term is found in the name fields, the email field, or the notes field:

```

SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'Morgan AND !Smith') > 0;
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'reuters') > 0;
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'Veronica') > 0;
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'435%5555') > 0;
-- queries can still be targeted at the individual fields in the indexed virtual document
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'reuters WITHIN email_addr') > 0;
SELECT c.* FROM customer c WHERE CONTAINS(otx_upd_flg,'asleep WITHIN note') > 0;
SELECT c.* FROM customer c WHERE CONTAINS(c.otx_upd_flg,'waived') > 0;
-- Shows that changes to a member of the multi-table index are immediately reflected
-- in CONTAINS queries, using the TRANSACTIONAL parameter, even though the index
-- has not been resync'd yet.
DELETE FROM customer_contact WHERE cust_id = 4;
SELECT c.* FROM customer c WHERE CONTAINS(c.otx_upd_flg,'waived') > 0;

```

New with 11g

11g introduced a host of new features and improvements, most of which are oriented toward more uptime, ease and performance of maintenance operations, none of which are the focus of this paper. One nice addition, though, is the new Oracle Text Manager portion of Oracle Enterprise Manager. For more information, turn to the 11g Oracle Text Reference manual, first section after the preface, “What’s New in Oracle Text?”

Conclusion

Cracking open the lid on Oracle Text feels like opening Pandora’s Box to some, but it doesn’t have to be. Yes, it is a huge and complicated product, but there are also features that are easy to access and use starting today. Now that you have been introduced, make your business aware of Text’s offerings and abilities. Could be your day to be the hero.

Appendix: Creating the EDW User

To follow the simple examples in this paper, you may want to create the following account, table, function and data in order to follow along with the scripts and code snippets in this paper.

```

CREATE USER edw IDENTIFIED BY "edw" DEFAULT TABLESPACE users TEMPORARY TABLESPACE temp
  PROFILE default QUOTA UNLIMITED ON users;
GRANT EXECUTE ON ctxsys.ctx_ddl TO edw;
GRANT CTXAPP TO edw;
GRANT CREATE PROCEDURE TO edw;
GRANT CREATE SESSION TO edw;
GRANT CREATE TABLE TO edw;
GRANT SELECT ANY DICTIONARY TO edw;
GRANT CREATE JOB TO edw;
GRANT CREATE TRIGGER TO edw;
CONNECT edw/edw

```

```
CREATE SYNONYM ctx_ddl FOR ctxsys.ddl;
```

DDL for PLACES table (simplified and denormalized for this example):

```
CREATE TABLE PLACES
(
  PLACE_ID          NUMBER(10) NOT NULL,
  PLACE_NM         VARCHAR2(60 CHAR) NOT NULL,
  PLACE_TYPE_NM    VARCHAR2(30 CHAR),
  PARENT_PLACE_ID NUMBER(10) REFERENCES places (place_id),
  PLACE_NOTES      CLOB,
  CONSTRAINT places_pk PRIMARY KEY (place_id),
  CONSTRAINT places_uk UNIQUE (place_nm, place_type_nm, parent_place_id)
);
```

Function so we can fill the place_notes column with something useful (tested on 10g):

```
CREATE OR REPLACE FUNCTION get_wiki(i_wiki_page IN VARCHAR2) RETURN CLOB
-- Borrowed from Dr. Tim Hall's excellent site Oracle-Base.com
AS
  l_http_request utl_http.req;
  l_http_response utl_http.resp;
  l_clob          CLOB;
  l_text         VARCHAR2(32767);
BEGIN
  dbms_lob.createtemporary(l_clob, FALSE);
  l_http_request := utl_http.begin_request('http://en.wikipedia.org/wiki/'||i_wiki_page);
  l_http_response := utl_http.get_response(l_http_request);
  BEGIN
    LOOP
      utl_http.read_text(l_http_response, l_text, 32767);
      dbms_lob.writeappend(l_clob, LENGTH(l_text), l_text);
    END LOOP;
  EXCEPTION
    WHEN utl_http.end_of_body THEN
      utl_http.end_response(l_http_response);
  END;
  RETURN l_clob;
EXCEPTION
  WHEN OTHERS THEN
    utl_http.end_response(l_http_response);
    dbms_lob.freetemporary(l_clob);
    RAISE;
END get_wiki;
```

Finally, some manufactured sample data for the PLACES table:

```
INSERT INTO places VALUES (251, 'United States', 'Country', NULL, get_wiki('UnitedStates'));
INSERT INTO places VALUES (35, 'Ohio', 'State', 251, get_wiki('Ohio'));
INSERT INTO places VALUES (1011, 'Toledo', 'City', 35, get_wiki('Toledo, Ohio'));
INSERT INTO places VALUES (51, 'District of Columbia', 'Political District', 251,
get_wiki('District_of_Columbia'));
INSERT INTO places VALUES (270, 'Northern Mariana Islands', 'Territory', 251,
get_wiki('Northern Mariana Islands'));
INSERT INTO places VALUES (231, 'Spain', 'Country', NULL, get_wiki('Spain'));
INSERT INTO places VALUES (3260, 'Málaga', 'Province', 231, get_wiki('Malaga'));
INSERT INTO places VALUES (3273, 'Toledo', 'Province', 231, get_wiki('Toledo, Spain'));
INSERT INTO places VALUES (146, 'Iceland', 'Country', NULL, get_wiki('Iceland'));
INSERT INTO places VALUES (3839, 'Höfuðborgarsvæði', 'Political District', 146,
get_wiki('Greater_Reykjavík_Area'));
INSERT INTO places VALUES (236, 'Sweden', 'Country', NULL, get_wiki('Sweden'));
INSERT INTO places VALUES (3393, 'Skåne', 'Province', 236, get_wiki('Scania'));
INSERT INTO places VALUES (997, 'Spam', 'Country', NULL, get_wiki('Spam_(electronic)'));
INSERT INTO places VALUES (998, 'Spam-Spam_and_Bacon', 'Province', 997, get_wiki('Spam_(food)'));
INSERT INTO places VALUES (999, 'spam egg spam spam bacon and spam', 'Township', 998,
get_wiki('Spam_(Monty_Python)'));
COMMIT;
```

Note: For the above inserts to work on 11g, the access to the 'net' must be granted so EDW can use the UTL packages. You can do so as SYS with something like this:

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.CREATE_ACL('users.xml',
    'ACL that lets users use the UTL packages.',
    'EDW', TRUE, 'connect');
END;
/

BEGIN
  DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE('/sys/acls/users.xml', 'EDW', TRUE, 'resolve');
END;
/
```