

More Oracle Text Tips

Douglas Scherer, Core Paradigm

Introduction

Oracle 8.1.7 provides Oracle Text users with advances in functionality, ease of use, and performance. One of its important components is a new Oracle Text feature, the CTXCAT type index (also know as *catalog* index). Oracle8i Release 3 (8.1.7) introduced the CTXCAT type index, adding features such as transactional synchronization of an Oracle Text index and a more "web-like" set of search operators. The CTXCAT type index also supports the use of *mixed queries*, structured criteria such as ORDER BY used in a CATSEARCH (the operator used when querying with a CTXCAT type index). Proper use of mixed queries will provide increased query performance.

Quick Review of Oracle Text CONTEXT Type Indexes

To help understand CTXCAT type indexes, we'll compare them against a know Oracle Text index type - CONTEXT indexes. CONTEXT indexes carry the key features described in this excerpt from an Oracle white paper of 2001:

- Indexes any document or textual content to add fast, accurate retrieval of information to Internet content management applications, eBusiness catalogs, news services, job postings, etc.
- Adds powerful text search and intelligent text management to Oracle 9i
- Fully integrated with Oracle 9i
- Offers premier text search quality
- Contains several advanced features for text management, document services, and XML
- Has best internationalization set of features for multilingual test search applications

For these examples, we'll look at a recipes table, which takes the following shape:

```
SQL> DESC recipes
Name                               Null?      Type
-----
ID                                  NOT NULL  NUMBER
NAME                                NOT NULL  VARCHAR2(100)
PREP_TIME_MINUTES                   NUMBER
SERVINGS                             NUMBER
DESCRIPTION                          VARCHAR2(1000)
COOKING_INSTRUCTIONS                 CLOB
DISH_IMAGE                           ORDSYS.ORDIMAGE
CULINARY_REVIEW                      BLOB
```

The primary key is built on ID. There are CONTEXT type indexes on NAME and on CULINARY_REVIEW. CULINARY_REVIEW contains Adobe Acrobat and MS Word files that reviewers have submitted. With that we can take a quick look at what's going on under the hood.

The columns we're most interested in these examples are ID and NAME. Here is the content of those columns:

ID NAME

```

-----
1 CB's Bean and Rice Soup
2 MC's tofu and rice surprise
3 Spanish Rice and Vegetable Stew

```

You can create a CONTEXT type index on NAME with the following command.

```

CREATE INDEX recipes_name_ix
ON recipes (name)
INDEXTYPE IS CTXSYS.CONTEXT;

```

CONTEXT type indexes are domain indexes. Domain indexes are user-defined. So a CONTEXT type index is a user-defined index that's been pre-defined or "built-in" by the Oracle developers. When you create the index with the statement above, there is more than one object created. Table 1 below shows all of the objects that are created after issuing the statement. In the background of the recipes_name_ix index there are four tables, each starting with DR\$(index_name). The table ending with \$I is used for storing the index data. There are additional objects created that have names beginning with SYS. These are created in the background in support of the index-only tables (IOTs) and LOBs of the DR\$ tables. Finally, there is an index object created with the name specified in the CREATE INDEX statement (in this case, recipes_name_ix).

Name	Type	Description
DR\$RECIPES_NAME_IX\$I	Table	Stores the index data (including the word and occurrence lists)
DR\$RECIPES_NAME_IX\$K	IOT Table	Stores information that translates ROWIDs and DOCIDs
DR\$RECIPES_NAME_IX\$N	IOT Table	Stores obsolete DOCIDs, scheduled for garbage collection during optimization
DR\$RECIPES_NAME_IX\$R	Table	Stores information used to translate DOCIDs to ROWIDs for CONTAINS queries
DR\$RECIPES_NAME_IX\$X	BTREE INDEX	BTREE Index on DR\$RECIPES_NAME_IX\$I (TOKEN_TEXT, TOKEN_TYPE, TOKEN_FIRST, TOKEN_LAST, TOKEN_COUNT)
SYS_IOT_TOP_34882	IOT Index	On DR\$RECIPES_NAME_IX\$K (TEXTKEY)
SYS_IOT_TOP_34887	IOT Index	On DR\$RECIPES_NAME_IX\$N (NLT_DOCID)
SYS_IL0000034879C00006\$\$	LOB Index	On DR\$RECIPES_NAME_IX\$I (TOKEN_INFO)
SYS_IL0000034884C00002\$\$	LOB Index	On DR\$RECIPES_NAME_IX\$R (DATA)
SYS_LOB0000034879C00006\$\$	LOB SEGMENT	For DR\$RECIPES_NAME_IX\$I.TOKEN_INFO

SYS_LOB0000034884C00002\$\$	LOB SEGMENT	For DR\$RECIPES_NAME_IX\$.DATA
RECIPES_NAME_IX	Domain Index	On RECIPES (Name)
TABLE 1: List of Objects Created for the Recipes_Name_IX Context Type Index		

CONTEXT type indexes use a predefined operator named "CONTAINS." A simple CONTAINS search looks like this:

```
SELECT id, name
   FROM recipes
  WHERE CONTAINS(name, 'rice') > 0;
```

Catalog Type Indexes

The CTXCAT type index—introduced in *Oracle8i Release 3*—is perfect for indexes like RECIPES_NAME_IX. That is, the CTXCAT type index works well with text fragments that are stored in VARCHAR2 or CLOB columns and don't need the rich set of CONTEXT document-handling features (themes, section searching, and so on). The following key features described CATALOG type index functionality.

- Good with text fragments
- Index sets supporting mixed queries
- Transactional synchronization of index and table data
- No document handling features
- No score value
- Web-like operators

DML changes to a CTXCAT type index are transactional. If a session performs a DML action against the NAME column in the recipes table, a query against NAME in the same session will recognize those changes, even if they have not yet issued a COMMIT (and as long as they have not been rolled-back). Other sessions will recognize the change immediately after the COMMIT. This is unlike the behavior of a CONTEXT type index, in which the index data must be periodically synchronized with its table's data. When using a CTXCAT type index, you are freed from the concern of Oracle Text index periodic synchronization.

An additional plus of the CTXCAT type index is its support of *mixed queries*. Mixed queries allow query criteria for other columns to be combined with the text search. They also allow for structured clauses (such as ORDER BY) to be included in the same index lookup as the text search itself. Mixed queries are made possible through *index sets* (objects that hold sets of ordered lists of columns from the indexed table), which are specified in the PARAMETERS clause of the CREATE INDEX statement.

Note that you can mix and match the CTXCAT and CONTEXT indexes — so you can create both types of index on the same text column.

Creating a CTXCAT type index

Try building a CTXCAT type index on NAME. First, drop the existing CONTEXT type index if it exists:

```
DROP INDEX recipes_name_ix;
```

The statement to build a CTXCAT type index is very similar to the one used to build a CONTEXT type index. Simply replace the INDEXTYPE specification with CTXSYS.CTXCAT as shown below:

```
CREATE INDEX recipes_name_ix
  ON recipes (name)
  INDEXTYPE IS CTXSYS.CTXCAT;
```

The CREATE INDEX statement can use a PARAMETERS clause, similar to the CREATE INDEX statement for a CONTEXT type index. The objects that are created after issuing this CREATE INDEX statement are shown below in Table 2.

<u>Name</u>	<u>Type</u>	<u>Description</u>
DR\$RECIPES_NAME_IX\$I	Table	Stores the index data (including the word and occurrence lists)
DR\$RECIPES_NAME_IX\$R	BTREE Index	BTREE Index on DR\$RECIPES_NAME_IX\$I (DR\$ROWID)
DR\$RECIPES_NAME_IX\$X	BTREE Index	BTREE Index on DR\$RECIPES_NAME_IX\$I (DR\$TOKEN, DR\$TOKEN_TYPE, DR\$ROWID)
DR\$RECIPES_NAME_IXTC	Trigger on recipes	Performs calls to CTXSYS owned procedure, which is used for synchronizing table changes with index set data
RECIPES_NAME_IX	Domain Index	Domain index on recipes (Name)
TABLE 2: List of Objects Created for the RECIPES_NAME_IX CTXCAT Type Index		

There are several important differences between the CTXCAT list in Table 2 and the CONTEXT list in Table 1:

- A trigger is created named DR\$RECIPES_NAME_IXTC
- The two IOT tables (DR\$RECIPES_NAME_IX\$K and DR\$RECIPES_NAME_IX\$N) are not created
- The object DR\$RECIPES_NAME_IX\$R is created as a BTREE index rather than a table
- There are no LOBs (following, there are no LOB segments or LOB indexes)

The first point shows a trigger, which is used for the synchronization of table data changes with index set data. The final three points are significant since they show a structural change in support of a new style of searching. It's a great storage boon that LOBs and IOTs are not created, but you should know that:

- The \$I table and its associated indexes contain more data with a CTXCAT type index than they do for a CONTEXT type index built on the same column.
- The column list of the \$I table for a CTXCAT type index will change depending on how you define your index sets.
- The use of index sets will increase the size of your \$I table.

In this example, no index sets are used.

To determine the amount of space used by 'DR\$RECIPES_NAME_IX%' CTXCAT type index, you can use the query below:

```
SELECT SUM(bytes)
  FROM user_segments
 WHERE segment_name LIKE 'DR$RECIPES_NAME_IX%';
```

Querying with a CTXCAT Type Index

The basic syntax for querying with a CTXCAT type index is similar in shape to the syntax used with a CONTEXT type index. A syntax difference is that a CONTEXT type index search uses the CONTAINS operator while a CTXCAT type index search uses the CATSEARCH operator. Look at the sample query below that returns rows from recipes where NAME contains the word "rice". Which rows do you think will be returned by the sample query below?

```
SELECT id, name
  FROM recipes
 WHERE CATSEARCH(name, 'rice', NULL) > 0;
```

The answer is that this query returns all of the rows in the recipes table. By default, a CTXCAT type index renders the search case insensitive (CONTEXT indexes have this same default). Every row contains the word "rice" although it appears in two forms: "rice" and "Rice".

The query language of the CATSEARCH operator is smaller, simpler and more "web-like" than that of the CONTAINS operator. This leads to easier application development. Below is a summarization of CATSEARCH query rules:

CATSEARCH query rules

- Multiple words are treated as AND
- A vertical bar is treated as OR
- A leading dash is treated as NOT
- Double quotes delimit phrases
- Parentheses group operations
- A plus sign in front of a word (used in some Web search engines) is ignored in a CATSEARCH query

Note: These are currently the only operators CATSEARCH supports. CONTEXT features (such as WITHIN and FUZZY searching) are not allowed.

CATSEARCH Operator Precedence

All of the operators can be used together to form complex queries; however, there is an order of

precedence, which is shown below:

1. Grouping ()
2. Phrase " "
3. NOT -
4. AND
5. OR |

The CATSEARCH operator takes three parameters in the following order:

1. The name of the indexed column
2. The search string
3. The reference to one or more index sets.

Remember, that the syntax for a CONTAINS search has an optional third parameter — a score label. Since the score is based on occurrences of a word within a document its document set, and the initial set under test, it is less meaningful in short text fragments. The third position of a CATSEARCH search references an index set, and this parameter in the CATSEARCH search must always be populated. So, if you don't have an index set you want to reference, just put "NULL" in the third position as shown in the above sample query.

OR

Words in the search string are ORed when separated by a vertical bar ("|"). The OR operator works the same whether or not there is white space before and/or after the vertical bar. Take the same search as above, but add a vertical bar between the words "rice" and "bean" as shown below.

```
SELECT id, name
FROM recipes
WHERE CATSEARCH(name, 'rice | bean', NULL) > 0;
```

This query will return all of the recipes since they all contain at least the word "rice".

AND

When several words in the search string are separated only by white space, they are ANDED. For example, consider the following query:

```
SELECT id, name
FROM recipes
WHERE CATSEARCH(name, 'rice bean', NULL) > 0;
```

Recipe 1 will be returned by this query since it contains the words "rice" and "bean".

NOT

NOT is specified with a dash ("-"). Here's the sample query again using the NOT operator:

```
SELECT id, name
FROM recipes
WHERE CATSEARCH(name, 'rice - bean', NULL) > 0;
```

Recipes 2 and 3 will be returned from this query. Recipe 1 is left out of the result set since it contains the

word "bean".

CATSEARCH does not allow a query in which all components in the search string are acted on by the NOT operator. So the following query returns the error, "DRG-50901: text query parser syntax error on line 1, column 1."

```
SELECT id, name
FROM recipes
WHERE CATSEARCH(name, '- rice - bean', NULL) > 0;
```

When there is no white space before and after the NOT operator, the words on either side of the operator are considered one term — as if they were concatenated. If we were to run the query shown below, which rows do you think would be returned?

```
SELECT id, name
FROM recipes
WHERE CATSEARCH(name, 'rice-bean', NULL) > 0;
```

The answer is that no rows would be returned from this query since the query would see the search string "ricebean." (Too bad such a recipe doesn't exist since it would probably have the most complete set of proteins that you could get in a query). This rule is used to facilitate ease of use. A clearer example is the hyphenation in the word "super-hero". The intention is not "super" NOT "hero", but something more akin to: "superhero".

A NOT operator is tolerated if it is positioned after white space and does not have trailing white space. The following query will return recipes 2 and 3:

```
SELECT id, name
FROM recipes
WHERE CATSEARCH(name, 'rice -bean', NULL) > 0;
```

Phrases

Double quotes (" ") delimit a phrase. Let's search for the phrase, "rice surprise":

```
SELECT id, name
FROM recipes
WHERE CATSEARCH(name, '"rice surprise"', NULL) > 0;
```

This query will return recipe 2.

Grouping in the search string

Words and phrases in the search string can be grouped in parentheses. The following query will return recipes with names that contain the words "rice" AND "tofu" OR names that contain the word "Spanish" (even if they don't contain both of the words "rice" AND "tofu").

```
SELECT id, name
FROM recipes
WHERE CATSEARCH(name, '(rice tofu) | spanish', NULL) > 0;
```

Recipes 2 and 3 will be in the result set.

Index Set Overview

The CTXCAT type index also supports the use of *mixed queries*, which are structured criteria (such as ORDER BY), used in a CATSEARCH (the operator used when querying with a CTXCAT type index). Proper use of mixed queries will provide increased query performance.

Mixed queries are made possible through the use of *index sets* (objects that hold sets of ordered lists of columns from the indexed table), which are specified in the PARAMETERS clause of a CREATE INDEX statement.

Index sets hold indexes and each of those indexes is an ordered list of base table columns for use in mixed queries. There are some examples using mixed queries in the section "Querying with Index Sets" below. First, however, let's go over the basic mechanics of creating index sets.

Index sets are defined using the CTX_DDL package. In these examples, the index set will be named RECIPES_ISET and will include two column lists (indexes): CALS and SERVINGS. There are three steps to preparing index sets:

1. Create the Index Set

Index sets are created using the CTX_DDL.CREATE_INDEX_SET package. CTX_DDL.CREATE_INDEX_SET takes one parameter: SET_NAME (VARCHAR2). Create the index set now with the following command:

```
SQL> EXEC CTX_DDL.CREATE_INDEX_SET('RECIPES_ISET')
```

2. Add Indexes to the Index Set

Indexes (column lists) are added using the CTX_DDL.ADD_INDEX procedure. CTX_DDL.ADD_INDEX takes two parameters: SET_NAME (VARCHAR2) and COLUMN_LIST (VARCHAR2).

Create two indexes using the following commands:

```
SQL> EXEC CTX_DDL.ADD_INDEX('RECIPES_ISET', 'CAL')
SQL> EXEC CTX_DDL.ADD_INDEX('RECIPES_ISET', 'SERVINGS')
```

You can see the indexes you've created in the CTX_USER_INDEX_SET_INDEXES view.

```
SQL> SELECT ixx_index_set_name,
           2   ixx_collist
           3   FROM ctx_user_index_set_indexes
           4   ORDER BY 1, 2;
```

```
IXX_INDEX_SET_NAME IXX_COLLIST
-----
RECIPES_ISET      CALS
RECIPES_ISET      SERVINGS
```

This query confirms that we have one index set (RECIPES_ISET) and two indexes in that index set (CAL and SERVINGS).

3. Create the CTXCAT Type Index Specifying the Index Set(s)

Finally, the index RECIPES_NAME_IX index needs to be created using the RECIPES_ISET index set.

If you created the index as CONTEXT type, you need to drop it before continuing. Unfortunately, index sets can't be added to existing indexes.

```
DROP INDEX recipes_name_ix;
```

Then create the index as in the following statement:

```
CREATE INDEX recipes_name_ix ON recipes (name)
  INDEXTYPE IS CTXSYS.CTXCAT
  PARAMETERS ('index set recipes_iset');
```

Using Index Sets

Now you're ready to issue mixed queries against the RECIPES table.

Simple

A simple CTXCAT search might look like this:

```
SELECT id, name
  FROM recipes
 WHERE CATSEARCH(name, 'rice',
                 NULL
                 ) > 0;
```

The Oracle server processes the CATSEARCH by querying the DR\$RECIPES_NAME_IX\$I table. That query is efficient since the internal query Oracle uses to perform this operation can find all of its data in the BTREE index DR\$RECIPES_NAME_IX\$X that was automatically created on DR\$RECIPES_NAME_IX\$I when the "CREATE INDEX RECIPES_NAME_IX ..." statement was issued.

ORDER BY

Imagine that you had not included index sets in the creation of the RECIPES_NAME_IX. If you wanted to perform an ORDER BY on the column CALS, you would add it outside of the CATSEARCH expression, as shown below:

```
SELECT id, name, cals
  FROM recipes
 WHERE CATSEARCH(name, 'rice',
                 NULL
                 ) > 0
 ORDER BY cals;
```

Now, take a look at what happens when you use index sets. Try the same query by including the structured criteria parameter:

```
SELECT id, name, cals
  FROM recipes
 WHERE CATSEARCH(name, 'rice',
                 'ORDER BY cals'
                 ) > 0;
```

Below are the executions plans or the query without and with the use of a mixed query and index sets. You can see by the execution plan the uses index sets that an additional sort is no longer needed to complete the query.

Execution Plan Without Mixed Query

Execution Plan

```
-----  
0  SELECT STATEMENT Optimizer=CHOOSE  
1 0  SORT (ORDER BY)  
2 1  TABLE ACCESS (BY INDEX ROWID) OF 'RECIPES'  
3 2  DOMAIN INDEX OF 'RECIPES_NAME_IX'
```

Execution Plan Without Mixed Query

Execution Plan

```
-----  
0  SELECT STATEMENT Optimizer=CHOOSE  
1 0  TABLE ACCESS (BY INDEX ROWID) OF 'RECIPES'  
2 1  DOMAIN INDEX OF 'RECIPES_NAME_IX'
```

Remember that the DR\$RECIPES_NAME_IX01 index was automatically built for you to support exactly this type of query, which includes a reference to CALS. The columns in DR\$RECIPES_NAME_IX01 contain the word and occurrence list *plus* the additional columns you specified in the column list for this index of the index set. This allows the query to find the word and process the ORDER BY with one index scan.

True, instead of using index sets, you could have built a separate BTREE index on the CALS column. But performing the query would have required a lookup on two indexes (the DR\$RECIPES_NAME_IX\$X and the second index you would have created on CALS) instead of just the former.

AND

Let's quickly look at some additional queries using index sets to give you a flavor of how your statements can take shape. All of these additional queries return recipe 2:

This:

```
SELECT id, name,  
       FROM recipes  
WHERE CATSEARCH(name, 'rice',  
                'cals <= 100  
                AND servings = 2'  
                ) > 0;
```

Versus:

```
SELECT id, name  
       FROM recipes
```

```
WHERE CATSEARCH(name, 'rice', NULL) > 0
      AND cals <= 100
      AND servings = 2;
```

B. Complex

This:

```
SELECT id, name, cals,
       servings
FROM recipes
WHERE CATSEARCH(name, 'rice',
                'cals IN (100, 300)
                AND servings = 2
                ORDER BY servings'
                ) > 0;
```

Versus:

```
SELECT id, name, cals,
       servings
FROM recipes
WHERE CATSEARCH(name, 'rice',
                'cals IN (100, 300)
                AND servings = 2'
                ) > 0
ORDER BY servings;
```

Rules that Govern the Use of Index Sets

Finally, there are rules that govern index sets and mixed queries.

Index Set Rules

- An index set can take up to ninety-nine indexes
- NULLs are not allowed in a column used in an index set index. NULLs will cause an index error and the row will not be indexed.
- The only allowed data types are: NUMBER, DATE, CHAR, and VARCHAR2
- The maximum length of a column in an index set's index is thirty bytes.

Mixed Query Rules

- The left-hand side (the column name) of the expression must be a column named in at least one of the indexes of the index set.
- The left-hand side must be a column name.
- The operators are limited to: <, <=, =, >=, >, BETWEEN, and IN.
- The right-hand side must be composed of literal values.
- Criteria can be combined with AND
- All of the columns in an ORDER BY must go in the same direction.

Conclusion

If you need to perform text searches on text fragments, want to perform mixed queries, would like transactional rather than periodic synchronization of your Oracle Text index with your table data, and can make use of a subset of CONTEXT index type query operators (and none of CONTEXT's document handling features), then you should use the CTXCAT type index.

About the Author

Douglas Scherer (dscherer@coreparadigm.com) is founder and president of Core Paradigm, a company that shows organizations how to realize the optimal value from their corporate knowledge. Mr. Scherer is a noted speaker and author on database analysis and design, implementation, and management. He is the lead author of *Oracle 8i Tips & Techniques*, and co-author of the *Oracle DBA Interactive Workbook* and *Oracle DBA Complete Video Course*. His Oracle Magazine series on interMedia and text mining is presented to over 500,000 readers annually.