

Exploring Oracle Text's CTXCAT Type Index

Learn about the CTXCAT type index and how it compares to the CONTEXT type index. / By Douglas Scherer

Oracle8i Release 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 known as the *catalog* index).

For comparison, let's look at a known index type from previous Oracle releases: the CONTEXT type index. I'll assume you've at least performed the test scenarios (or ones similar) as described in my previous article, "Exploring Oracle Text Basics" (March/April 2001). Listing 3 on the last page of this issue's article will refresh your memory of the RECIPES table and data used in those discussions. When the CONTEXT type index `recipes_name_ix` was created on the NAME column, more than one object was created. Table 1 shows all objects created after issuing the following statement:

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

CONTEXT type indexes are domain indexes, and domain indexes are user-defined. So a CONTEXT type index is a user-defined index that's been predefined or "built-in" by the Oracle developers. CONTEXT type indexes use a predefined operator named CONTAINS. A simple CONTAINS search looks like this:

Philip Brooker

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

In the background there are four tables, each starting with `DRS<index_name>` table ending with `$I` 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 DRS tables. Finally, there is an index object created with the name specified in the `CREATE INDEX` statement (in this case, `recipes_name_ix`).

You will not find any segments listed in the `USER_SEGMENTS` data-dictionary view for IOT `DRSRECIPES_NAME_IXSK` and `DRSRECIPES_NAME_IXSN` or the domain index `recipes_name_ix`. So, how can you find the total amount of storage used by the `recipes_name_ix` index? The query shown in Listing 1 provides you with that information. To see the space used by a different CONTEXT type index, substitute the name of that index for `recipes_name_ix`.

CONTEXT TYPE VERSUS CTXCAT 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).



See Douglas Scherer's March/April 2001 article, "Exploring Oracle Text Basics," online at www.oracle.com/oramag/.

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.

These are currently the only operators **CATSEARCH** supports. **CONTEXT** features such as **WITHIN** and **FUZZY** searching are not allowed.

The total size of a **CONTEXT** type index can range from 30 percent to 200 percent of the size of the indexed information.

DML changes to a **CTXCAT** type index are transactional. If a session performs a DML action against the **RECIPES** table, the query against **NAME** will recognize those changes, even if they have not yet issued a **COMMIT** (and as long as they have not been rolled back). This is unlike the behavior of a **CONTEXT** type index, for 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 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. Index sets will be discussed in more detail in the next article in this series.

BUILDING 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 for building a **CTXCAT** type index is very similar to the one for building 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;
```

Table 2 shows the objects that were created after this **CREATE INDEX** statement was issued. The **CREATE INDEX** statement can use a **PARAMETERS** clause that is similar to the **CREATE INDEX** statement for a **CONTEXT** type index. There are several

TABLE 1: LIST OF OBJECTS CREATED FOR THE RECIPES_NAME_IX CONTEXT TYPE INDEX

Name	Type	Description
DRSRECIPES_NAME_IXSI	Table	Stores the index data (including the word and occurrence lists)
DRSRECIPES_NAME_IXSK	IOT Table	Stores information that translates ROWIDs and DOCIDs
DRSRECIPES_NAME_IXSN	IOT Table	Stores obsolete DOCIDs , scheduled for garbage collection during optimization
DRSRECIPES_NAME_IXSR	Table	Stores information used to translate DOCIDs to ROWIDs for CONTAINS queries
DRSRECIPES_NAME_IXSX	BTREE INDEX	BTREE Index on DRSRECIPES_NAME_IXSI (TOKEN_TEXT , TOKEN_TYPE , TOKEN_FIRST , TOKEN_LAST , TOKEN_COUNT)
SYS_IOT_TOP_34882	IOT Index	On DRSRECIPES_NAME_IXSK (TEXTKEY)
SYS_IOT_TOP_34887	IOT Index	On DRSRECIPES_NAME_IXSN (NLT_DOCID)
SYS_IL0000034879C00006\$\$	LOB Index	On DRSRECIPES_NAME_IXSI (TOKEN_INFO)
SYS_IL0000034884C00002\$\$	LOB Index	On DRSRECIPES_NAME_IXSR (DATA)
SYS_LOB0000034879C00006\$\$	LOB SEGMENT	For DRSRECIPES_NAME_IXSI.TOKEN_INFO
SYS_LOB0000034884C00002\$\$	LOB SEGMENT	For DRSRECIPES_NAME_IXSR.DATA
RECIPES_NAME_IX	Domain Index	On RECIPES (Name)

LISTING 1

Query for finding the current size of the **DR\$RECIPES_NAME_IX%** CONTEXT type index. Note: To use this query for a different CONTEXT type index, plug in the name of the index where **RECIPES_NAME_IX** appears.

```
SELECT SUM(bytes)
  FROM user_segments
 WHERE segment_name IN
       (SELECT segment_name
        FROM user_lobs
         WHERE table_name LIKE 'DR$RECIPES_NAME_IX%'
        UNION ALL
        SELECT index_name
        FROM user_indexes
         WHERE table_name LIKE 'DR$RECIPES_NAME_IX%'
        UNION ALL
        SELECT table_name
        FROM user_tables
         WHERE table_name LIKE 'DR$RECIPES_NAME_IX%'
       );
```

important differences between the CTXCAT list in Table 2 and the CONTEXT list in Table 1. In Table 2

- ◆ A trigger named **DR\$RECIPES_NAME_I_XTC** is created.
- ◆ The IOT tables (**DR\$RECIPES_NAME_IXSK** and **DR\$RECIPES_NAME_IXSN**) are not created.
- ◆ The object **DR\$RECIPES_NAME_IXSR** 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 because they show a structural change in support of a new style of searching. Although it's a great storage boon that LOBs and IOTs are not created, you should keep the following in mind:

- ◆ 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.

You can use the query in Listing 2 to find the amount of space used by a CTXCAT type index. Although you can use the query in Listing 1, the Listing 2 query will run more efficiently. You cannot, however, use the query in Listing 2 to see storage usage of a CONTEXT type index.

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 whereas a CTXCAT type index search uses the **CATSEARCH** operator. Look at the sample query below, which returns rows from recipes where **NAME** contains the word *rice*. Listing 3 also displays the names in the **RECIPES** table. Which rows do you think will be returned by this query?

```
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

TABLE 2: LIST OF OBJECTS CREATED FOR THE RECIPES_NAME_IX CTXCAT TYPE INDEX

Name	Type	Description
DR\$RECIPES_NAME_IXSI	Table	Stores the index data (including the word and occurrence lists)
DR\$RECIPES_NAME_IXSR	BTREE Index	BTREE Index on DR\$RECIPES_NAME_IXSI (DR\$ROWID)
DR\$RECIPES_NAME_IXSX	BTREE Index	BTREE Index on DR\$RECIPES_NAME_IXSI (DR\$TOKEN , DR\$TOKEN_TYPE , DR\$ROWID)
DR\$RECIPES_NAME_I_XTC	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)

 For more information, go to otn.oracle.com/products/intermedia/ and then click on Oracle Text. There you'll find code samples, downloads, and the Oracle Documentation set. There's also a discussion forum at otn.oracle.com. From the home page, click on Discussions (in the left-hand panel) and then on Enter a Technical Forum. Look under Technologies for the Oracle Text discussion forum.

the word *rice*, although it appears in two forms: *rice* and *Rice*.

The CATSEARCH operator takes three parameters, in the following order:

- ◆ The name of the indexed column
- ◆ The search string
- ◆ 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 and its document set, it is less meaningful in short text fragments. The third position of a CATSEARCH search references an index set. The third parameter in the CATSEARCH search must always be populated. So, if you don't have an index set you want to reference, put NULL in the third position, as shown in the above sample query.

THE QUERY LANGUAGE

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. For a summary of the query language rules, see the “CATSEARCH Query Rules” sidebar.

A CATSEARCH search supports logical AND, OR, and NOT.

When several words in the search string are separated only by white space, they are ANDed. Consider the following query:

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

Recipe 1 will be returned by this query,

CATSEARCH OPERATOR PRECEDENCE

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

since it contains the words *rice* and *bean*.
OR

Words in the search string are ORed when separated by a vertical bar (|). The OR operator works in the same way 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*:

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

This query will return all the recipes, since they all contain at least the word *rice*.

NOT

NOT is specified with a dash (-). Here's the sample query again, this time 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 where 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;
```

LISTING 2

Query for finding the current size of the DR\$RECIPES_NAME_IX% CTXCAT type index. Note: To use this query for a different CTXCAT type index, plug in the name of the index where RECIPES_NAME_IX appears.

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

LISTING 3

When you last saw the RECIPES table, it had 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 is a CONTEXT type index on NAME, and you may have added a CONTEXT type index on CULINARY_REVIEW. CULINARY_REVIEW was added per a sidebar in a previous article. It contains Adobe Acrobat and MS Word files that reviewers have submitted.

The columns we're most interested in for this article 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

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 would be returned?

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

No rows will be returned from this query, since the query will see the search string, *ricebean*. 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 that is positioned after white space and does not have trailing white space is tolerated. 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*:

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

Recipes 2 and 3 will be in the result set.

You can use all the operators together to form complex queries. The “CATSEARCH Operator Precedence” sidebar shows the precedence of the operators.

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. ◆

Douglas Scherer (dscherer@coreparadigm.com) is president of Core Paradigm, a management consulting firm that helps C-level executives identify and fulfill their IT needs. He is a frequent presenter at international conferences and author of two books on Oracle technology. Scherer also teaches at Columbia University.