Basics of Oracle Text Retrieval

Paul Dixon
Oracle Corporation
paul.dixon@oracle.com

Abstract

This paper covers the basics of text search within the Oracle RDBMS. Emphasis is on demonstrating the simplicity and power of the SQL CONTAINS() text query language.

1 Introduction

For the average user, searching through documents in a file system means using "grep" - this is a linear scan using regular expressions. At a certain size of collection an index becomes necessary. For text this will typically be an "inverted list" index, although other text indexes may be used for specialized applications. An inverted list looks the same wherever the source documents may reside; in a database, on a file system, or on the Web. In Oracle, text indexes are stored natively as database tables.

The primary interface to the relational database is SQL (Structured Query Language), which most people know from handling structured data (e.g., numbers, dates, short text strings, etc.), and the interface to the text-retrieval component is no exception. The primary interface to text is the CONTAINS() function, but CONTAINS() works within the greater framework of the SQL language.

2 Defining the Repository

In a file system, a file has disk storage and metadata, whereas in a database a document consists of a database cell (or a set of cells), where a database cell occurs in the individual row and column of a table. A file will generally have an associated system date and other meta information will appear as a file name and security permissions. In the database, metadata is not predefined, but can include system date and arbitrary associated column data.

Database storage is defined in terms of database tables and types:

Example 1: CREATE TABLE recipes (name CHAR(30), directions VARCHAR2(400));

This is nearly the simplest possible schema definition (a 30-character-width text field and a 400-character-maximum variable-width field). It could be augmented with other types; structured types such as the date of insertion, the time to create the recipe, cross references, id number, and so on. Text itself can be stored in multiple formats - in a LOB (large object) or as a Binary LOB if it is in a non-text format, such as a word-processing file, or is in a character set other than that of the host database. Other supported datatypes include Web (URL), external file, and procedural (user-defined).

Copyright 2001 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

3 Getting Documents In

The SQL INSERT statement is the most accessible and familiar interface for database data input.

Example 2: INSERT INTO recipes VALUES ('omelet', 'Finely chop onion, put 1 tbsp. butter in skillet over high flame, mix two eggs, salt and pepper with fork in a bowl, add to skillet, shake and run fork through eggs, when eggs are set, add cheese and onion, fold in half, serve.');

INSERT is primarily designed for shorter text fragments. Other methods of inserting documents are the Oracle Call Interface (C language), SQL*Loader (a stand-alone specialized data/document loader), and the Oracle Internet File System, which supports file system protocols, as well as FTP and HTTP.

4 Finding Information

Creating a full-text index is straightforward:

Example 3: CREATE INDEX recipeidx ON recipes(directions) INDEXTYPE IS CONTEXT;

This create index statement works like a "normal" B-tree index. It creates a set of tables that comprise an inverted list index on the text column. The inverted index is effectively the same as the back-of-book index in a paper book; it matches a token or phrase and its locations to speed up search with a look-up rather than a scan.

To query the text column, one uses CONTAINS:

Example 4: SELECT name FROM recipes WHERE CONTAINS(directions, 'cheese')>0;

Here the SELECT acts upon the full-text index, to return all entries that contain the token 'cheese' within the 'directions' field. The CONTAINS function can be included in SQL of arbitrary complexity, in conjunction with structured conditions, join conditions, views and sub-selects; it is a row source native to the Oracle RDBMS. When select statements include both full-text and structured components, the optimizer will choose the correct execution plan based on the cost and selectivity of all predicates. It is possible to influence the plan generation for text using optimizer "hints", the same as for structured conditions.

5 Updating the Collection

To add to the indexed document collection, we can insert new documents:

Example 5: INSERT INTO recipes VALUES('grilled cheese sandwich', 'oil skillet and heat over medium flame, put one slice of cheddar cheese and one of cheese (jack) between bread slices, fry on each side until brown');

Unlike Oracle B-tree indexes, text indexes need to be explicitly updated. This is to avoid index fragmentation due to the relatively small number of individual token occurrences within a single document.

Example 6: ALTER INDEX recipeidx REBUILD ONLINE PARAMETERS('SYNC');

Index synchronization is usually run as a timed job (DBMS_JOB) at an interval based upon the degree of index fragmentation, and the document DML rates. Where batch-inserts are the norm, synchronization will take place immediately following, whereas in an ad-hoc insert environment, SYNC is usually run at regular intervals throughout the day.

6 Search Techniques

In its simplest form, CONTAINS can be thought of as a boolean function (ignoring the integer return value). For a token or phrase ("keyword") search this can be perfectly acceptable, particularly on small document collections, or short documents (such as would be found in a product catalog). For larger sets, however, SCORE() can be used to rank the result-set by degree of relevance to the text query.

CONTAINS has its own internal extended boolean text-query syntax, which can be used to augment the text tokens searched for. Below are a few of the most useful features.

6.1 Basic

Relevance, for the context index, is defined in terms of term frequency (the more times a term occurs in a document, the higher the relevance), dampened by the total number of occurrences of the term in the collection. If a term is frequent in a document collection, we say it contributes less information content - it is more 'noise'.

The simplest ordered query is on a single token:

Example 7: SELECT name, SCORE(1) FROM recipes WHERE CONTAINS(directions, 'cheese', 1)>0 ORDER BY SCORE(1) DESC;

NAME	SCORE(1)
grilled cheese sandwich	7
omelet	4

Here we can see that the document that mentions 'cheese' more often (twice) gets the higher relevance-ranking. SCORE() is a pseudo-column that carries the relevance information ancillary to the CONTAINS function. The label, '1' in this case, is used to match the SCORE to the appropriate CONTAINS, as there may be several CONTAINS in a single SELECT statement.

6.2 Weight (*)

Each query component can be differentiated by 'weighting': for instance the top criterion may be be weighted four times as much as the second, which is weighted twice the weight of the third, in order to give a progressive relaxation of subqueries.

Example 8: 'onion * 4 AND cheddar'

Here the 'onion' part of the text query is weighted higher than the 'cheddar' component. All documents that match this query will have both terms, but SCORE will be higher for 'onion' unless it is greatly outnumbered by the other term.

6.3 Phrase

Phrase searches find an exact match. 'cheddar cheese' finds only those two words in sequence and next to each other.

6.4 NEAR (;)

NEAR is similar to 'phrase' but tokens only need to be close to each other, not next to each other - it is a kind of less-severe proximity search. Words can appear in any order. Similar to phrase, NEAR has a linguistic aspect to it, which is loosely defined; such a definition would be something like "two words occurring in the same sentence" or "two words occurring in the same paragraph."

Example 9: 'jack ; cheese'

This query finds 'cheese (jack),' which would be missed by a phrase-only search. It would, however, find 'jack and jill, mouse cheese,' which might not be desired.

6.5 Accumulate

Accumulate always assigns a higher SCORE to a document matching any N query components than to a document matching only any $N-1$ components. Even though all components may not occur in a document, however, SCORE will be positive if individual terms do occur. This gives high query relevance at the top of a score-sorted result set, while allowing a longer 'tail', to reduce the chances of an empty result.

Example 10: 'cheese , swiss , cheddar'

This query will match both example documents, even though only two of the terms are found. The 'grilled cheese sandwich' document, matching two query terms, will score higher than the 'omelet' document, which matches only one term.

7 Further Topics

Other text query operators available to the out-of-the-box Oracle Text installation are boolean AND, OR, NOT, and MINUS, a fuzzy interface to find spelling and OCR errors, and a complete thesaurus infrastructure. Section searching is available to exploit document structure in HTML or XML marked-up documents. A highlighting interface marks up the query terms where they appear in a document, and converts popular word-processor formats to HTML.

Oracle Text can also produce a 'theme index'[1]. This is an inverted-list index of concepts derived from the document collection and a 425,000-term knowledge base. The system has also been used to build automatic classification and question-answering systems[2].

References

[1] K. Mahesh, J. Kud, and P. Dixon. *Oracle at TREC8: A Lexical Approach*. Proceedings of the Eighth Text Retrieval Conference, 1999.

[2] S. Alpha, P. Dixon, C. Liao, and C. Yang. *Oracle at TREC10.* Notebook paper, 2001; available at `http://trec.nist.gov/pubs/trec10/papers/orcltrec10.pdf`.