Précis Index Implementation for Efficient Fulltext Data Mining

Michal Kopecký[1] and Martin Čermák

Department of Software Engineering,
[1]Faculty of Mathematics and Physics, Charles University Prague
Malostranské nám. 25, 118 00 Prague, Czech Republic
michal.kopecky@mff.cuni.cz, cermak23@gmail.com

Abstract. Précis system has been designed for text based searching over relational database. Unlike the common approach, this system allows user to search requested data over a whole database, not only within one table. System takes queries formulated in free-form and produces rows containing information corresponding to the query and also information associated to them within the database. This paper presents implementation of the index, suitable for efficient searching in the Oracle relational database management system. Implementation provides SQL query language extension for searching desired data.

Keywords: Information Retrieval, Précis index, Relational Databases, Data Mining.

1 Introduction

In the relational databases application designers have to define appropriate set of indexes to speed-up the searching process and data manipulation. These indexes are usually implemented using redundant B^+ trees. It is possible to find also different index type implementations recently. Some of them try to overcome B+ tree limitations as requirement for high selectivity while other try to support searching completely different types of data (XML, images, texts, geographical data etc.).

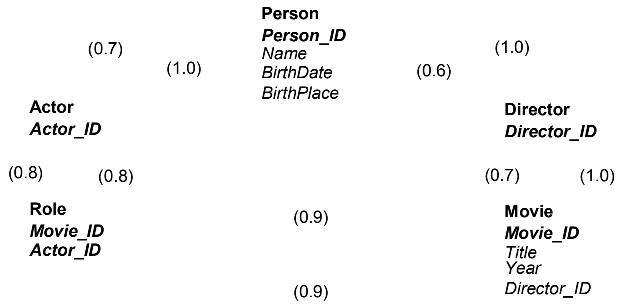
Standard searching within the database requires that application developers understand the database structure. The Précis system invented in [1] allows users to query the database without exact knowledge about the database and provides them with exhaustive answer. Users formulate queries using keywords and the system looks for all relevant data within the database independently on the fact in which table and/or column the information are stored. Moreover the system is able to trace foreign key – primary key links within the database and find out also related data. The result can be then presented as a hierarchy of resulting rows, or can be formatted to sentences in natural language.

However, the existing built-in indexes doesn't support search over all columns of all tables and naïve implementation of the system using LIKE operator will be extremely slow and ineffective. This paper presents the implementation of the Précis index in the Oracle database system using its extensibility features.

Following chapter describes the Précis indexing in more details. The paper concerns on brief implementation description and obtained performance results. More details about the implementation can be found in [2]

2 Précis system

The primary data are stored in classical database and can be processed by standard database application. The Précis querying uses data stored in all tables and columns and takes into account associations between tables derived from foreign key references present in the database. References, tables and columns in the database can be additionally rated by weights, represented by numbers $w \in <0;1>$. These weights can be either global or associated to individual users. The Précis query q consists of a set of keywords $q=\{k_1, k_2, \ldots, k_n\}$. According to the given query the system finds out corresponding records and related information. Let suppose following database structure:



Picture 1. Sample database structure

The broader arrows represent foreign keys in the database, narrower dotted arrows represent opposite associations. Numbers in square brackets represent ratings for associations. Having such a database, the query q={"*Woody*", "*Allen*"} could provide the result (without text formatting) in form:

```
Director: Woody Allen | December 1st 1935 | Brooklyn | New York | USA
Movie:    Match Point                | 2005
Movie:    Melinda and Melinda        | 2004
Movie:    Anything Else              | 2003
Role:     Hollywood Ending                          | 2002
Role:     The Curse of the Jade Scorpion            | 2001
```

Authors of the Précis system have proposed the query evaluation is in a sequence of four successive steps: ***First*** – create list of all occurrences of all query terms in the database D. This step provides subschema D' of all tables containing those terms. ***Second*** – determine the subset of the database schema accessible from D' using associations. This step provides subschema D”. ***Third*** – fill database **D**” with the schema D” with retrieved data. ***Last*** – optionally transform data in **D**” to the natural language using text templates.

Each record in the result database obtains a rating according to its relation to data found in the first step. The rating of the row accessible through a chain of association is computed as a product of ratings assigned to corresponding tables, columns and association. The result then contains all data with rating exceeding given threshold.

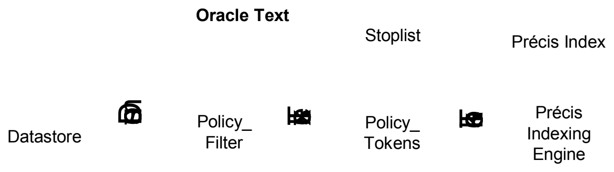
3 Implementation

Our concern was on efficient inverted index building in one of most used RDBMS in the enterprise segment – the Oracle database – and on query evaluation using this index. The Oracle database was chosen because it provides developers with necessary background for creating user-defined index types.

The basic requirements on the Précis index were as follows:

- Each index should contain a set of columns stored in arbitrary number of tables within the database.
- Each user should be able to create any number of indexes.
- One column can be assigned to any number of defined indexes.
- The user interface for index manipulation should be as user-friendly as possible Querying the database should be available through the Oracle SQL query language.

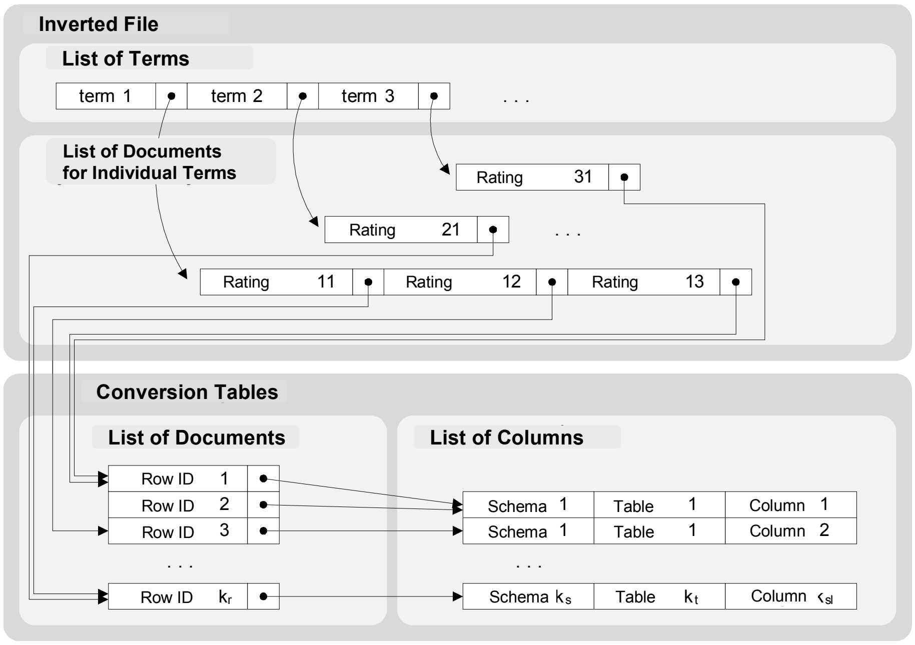
According to above stated requirements, it is necessary to consider each value stored in any of indexed columns as a separate textual document.
The Précis index creation process is shown on picture 2.



Picture 2. Précis index creation process

Each document obtains its unique identifier based on the table owner, table name, column name and row within the database. Document stored in binary format are filtered and converted to plaintext. Text is tokenized and converted to a set of terms together with number of their occurrences within the document. Filtering and tokenization uses functionality available in Oracle Media Text extension [3]. For each term its *term frequency* (TF) within a given document document and inverted document frequency (IDF) is computed and the standard TF*IDF formulae [4] is then used for term weight computation.

Logical structure of the Précis inverted index is shown on picture 3.



Picture 3. Logical structure of Précis index

Data are stored within BLOB records, divided to blocks of fixed size. Document lists short enough to not exceed one block, are stored directly in the block ordered by document ID. Longer lists are stored in form of non-redundant B-trees. Each BLOB record can contain blocks of the same type. It is possible to use more BLOB records containing ordered lists, providing that their block sizes differ each to other. Having more available blocksizes ranging from ones to hundreds items speeds up both searching and index synchronization.

First block in each BLOB – the header – contains metadata about the BLOB contents. The most of its content represents a beginning of free blocks list. The list continues after the last used block. During synchronization the free block list is stored in the core memory and is written back afterwards.

Position	**Size**	**Description**
0	2B	Number of used blocks in the record
2B	4B	Number of block with the rest of free records.
6B	4B	Number of blocks used for free blocks list.
10+	4B	Numbers of first 1000 free blocks within the BLOB.

Table 1: BLOB header structure

B-tree nodes in the BLOB records have similar format. Two bytes at the beginning of the node contain number of items stored within the node. Then follow pointers interlieved with item records. Item records contain the key and needed metadata. Metadata can either contain list of documents ordered by its numbers or its weights or it can contain lists of terms with lengths ranging from 1-6, 7-12, 13-24, respectively 25-64 characters. Details are shown in following tables.

B-tree purpose	**Field**	**Overal Size**	**Size**	**Decription**
Terms	Key	6, 12, 24, 64	6, 12, 24, 64	Term padded with trailing zero-bytes
	Metadata	12	2	Number of docs containing term
			2	Number of BLOB record with document list
			4	Number of block within the BLOB record containing list ordered by document number.
			4	Number of block within the BLOB record containing list ordered by term weigth. Zero, if the previous list is not stored as B-tree
Documents ordered by number	Key	4	4	Document number
	Metadata	2	2	Document weight
Documents ordered by weigth	Key	6	2	Document weight
			4	Document number
	Metadata	-	-	-

Table 2: B-tree item record structure

3.1 Programming Language

The application is split to two layers. Upper layer is written in PL/SQL language and contains procedures, monitoring database changes and the user interface. Lower layer is written in C++ and maintains internal index structures. Communication between layers is implemented thouigh OCI interface. The C++ performace is much higher than the alternative PL/SQL implementation. On the other hand, the invocation of procedures written in C++ from the PL/SQL code requires substantial overhead. The code was therefore written to minimize switches from PL/SQL to C++ as much as possible.

3.2 User Interface

The user wanting use Précis index has to have assigned role PRECISAPP containing the role CTXAPP that allows usage of Oracle Media Text features. To maintain indexes including foreign columns, the user has to posses additional privileges CREATE ANY TRIGGER, ADMINISTER DATABASE TRIGGER and SELECT on given TABLE. Due to security risks maintaining foreign columns is initially disallowed.

Index maintainance is done through package PRECIS with following interface.

Function/Procedure	**Description**
CREATE_INDEX	Creates index, auxiliary tables and needed triggers.
DROP_INDEX	Drops the index including all data.
DROP_INDEX_FORCE	The same as above, without existency checks.
ADD_COLUMN	Adds new table column to the index.
DROP_COLUMN	Removes the table column from the index.
SYNCHRONIZE	Indexes data added and/or changed from previous synchronization.
SET_AS_SINGLE_SCHEMA SET_AS_ MULTI_SCHEMA	Sets, if it is possible to add columns belonging to tables from other schemas.

Table 3: PRECIS package interface

3.3 Index Search

The search is accelerated through the domain index built upon column DOC_ID of table PRECIS$*index_name*$D. This table formally holds all documents, physically stored in their repective tables. The query searches data from this table or any view created over this table using operator PRECISSYS.CONTAINS. The system provides the view PRECIS$*index_name*$S that provides additional columns stored in internal index tables. The query would look like

```
SELECT S.*, PRECISSYS.SCORE(n)
  FROM PRECIS$index_name$S S
  WHERE PRECISSYS.CONTAINS(DOC_ID,Q,L,n) > Treshold
```

where *Q* holds Boolean expression with keywords, *L* means limit – maximal required number of hits and a *Treshold* keeps minimal required document rating. Last parameter *n* holds numerical identifier of operator within the SELECT statement and allows obtaining of assigned document rating using auxiliary operator SCORE with the corresponding identifier. Using Boolean expressions instead of simple keyword extends the original proposition. To formulate queries, users can use brackets, and logical operators AND "&", OR "|" and NOT "-". The structure of view PRECIS$*index_name*$S is described in following table. The column TEXT returns the beginning of the document and is present for testing purposes.

During evaluation, terms used in the query are identified in the index. If the length of the term exceeds given limit suffix_search_min_token_len, the index searches for all its rightwise extensions. Shorter terms are searched in their exact form.

Query evaluation is based on Paice model [5]. Index search then identifies all documents that have high weight assigned to at least one term used in the query. To identify them lists ordered by weight in descendant order by weight are used. The exact rating for these documents is then evaluated using lists ordered by document numbers.

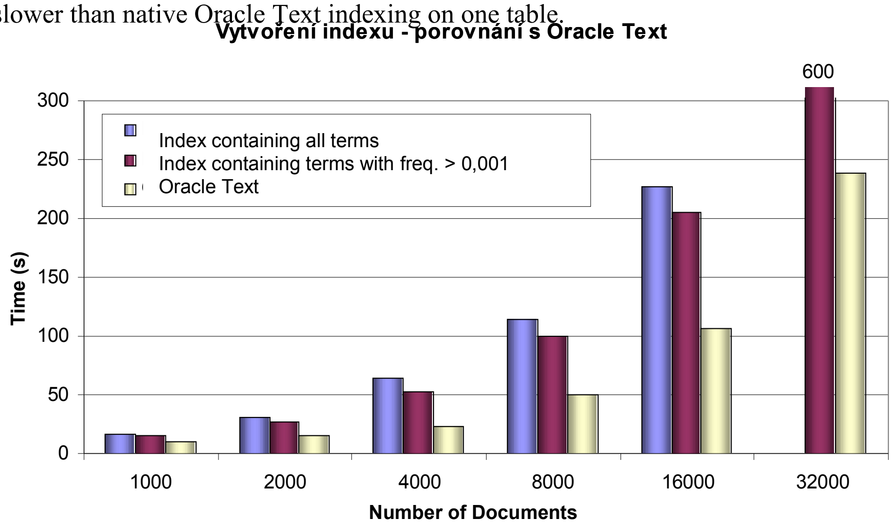
3.4 Index Data modification

Indexes are synchronized in batches. Changes are logged in the table PRECISSYS.PRECIS_PENDING. To keep track of changes Précis index uses two types of triggers. One of them tracks DML changes in tables and the second one watches for DROP TABLE and TRUNCATE TABLE statements. The INSERT OR UPDATE OR DELETE FOR EACH ROW trigger has to be created for each indexed table. Its name is PRECIS_*IndexID_TableID*_TRG. The trigger PRECIS_TBLDDL_TRG of the second type is created once in each schema containing at least one indexed column. Each document has assigned its unique identifier within the index, that can be translated to the quadruple (owner, table_name, column_name, rowid) through tables PRECIS$*index_name*$D and PRECIS$*index_name*$U. All BLOB records are stored in the table PRECIS$*index_name*$I. Parametrs of all precise indexes are stored in the table PRECISSYS.PRECIS_PARAMETER.

4 Performance tests

The graph on picture 4 shows result for index creation in comparison with *Oracle Multimedia Text*.

The time for 32000 documents increases significantly due to insufficient memory for buffers, so the data have to be transferred to a from the disk and the overall number of I/O operation was increased. In average, the index creation process is 2-3 times slower than native Oracle Text indexing on one table.



Picture 4. Précis index creation

The query evaluation was tested on collection containing 32000 documents from Wikipedia. Results represent execution times for queries Q_1 = "Astronomy", Q_2 = "Information System", Q_3 = "Pulp Fiction", Q_4 = "Database Search Oracle MySQL", Q_5 = "(Information Retrieval | Text Retrieval Systems)", Q_6 = "Relational Database Index".

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6
Rating computation	0.029	0.032	0.058	0.053	0.056	0.173
Data retrieval	0.052	0.163	0.205	0.259	0.241	0.430
Total	0.081	0.195	0.263	0.312	0.297	0.603

Table 4: Index search results

5 Conclusion

We proposed data structures suitable for Précis indexing and implemented it in the widespread Oracle database. The proposed implementation extends original idea by possibility to formulate queries using Boolean operators. The formulation of queries is familiar to Oracle application developers because of its similarity to Oracle Multimedia Text queries. Embeding most of document processing path provided by Oracle Media Text allows maintaining not only data in plain text columns, but also documents in different format stored in the database. The proposed data structures still provide quick searching of documents together with tolerable speed of indexing process.

References

1. Simitsis A. and col.: Précis: from unstructured keywords as queries to structured databases as answers. International Journal on Very Large Data Bases (VLDB Journal), Volume 17, Number 1, 2008, 117-149.
2. Čermák M.: Master Thesis: Index pro textové vyhledávání nad relačními daty, Charles University Prague, 2008. (in Czech language)
3. Oracle Text: Application Developer's Guide, 10g Release 2, 2005.
4. Salton, G. and M. J. McGill (1983). *Introduction to modern information retrieval.* McGraw-Hill. ISBN 0070544840.
5. Paice, C. P. (1984), *Soft Evaluation of Boolean Search Queries in Information Retrieval Systems*, Information Technology, Res. Dev. Applications, 3(1), 33-42