Toward Exhaustive Teaching Of Relational Database Indexes

Massimo MACERA¹, Paolo ROCCHI ²
¹IBM, via Sciangai 53 Rome 00144, Italy
²LUISS University, via Alberoni 7 Rome, Italy

Abstract - There are several indexing techniques that can optimize system performances. The choice of the right index in a relational database can improve the performances by up to 80%. Unfortunately, the illustrations of the various techniques are scattered over a number of texts and manuals, and the courses for database designers are often somewhat incomplete. The present paper shows a didactical experience which tackles this kind of limitations. In particular, we have prepared a textbook that includes twenty-one different index formats, it discusses advantages and disadvantages of each indexing technique, and has been positively validated during advanced courses on relational database design.

Keywords: Database index design, DB Indexing

1 Introduction

Ricardo and others examine challenging aspects of modern education on database (DB) systems [1] in the light of CC2001 recommendations [2]. These authors are inclined to focus on introductory lessons [3], but significant defects emerge even in advanced database courses which appear insufficient to satisfy the requirements of professional practice. Some educational weaknesses appear evident respect to working environment. As first, a number of courses on database systems concentrate on abstract principles and theoretical topics. Relatively little mathematics turns out to be important for software engineers in practice and it tends to be forgotten. As second, teachers tend to use rather simple, well-analyzed and well-understood datasets as both examples and project data in their courses [4]. Topics covered in undergraduate and graduate courses are often simplified because it is too difficult to teach a topic with all the complexity of the real world [5]. At the other hand young people, trained in simpler domains, are not adequately prepared for the actual assignments they will be facing in their future works [6]. Students will find much more complex database systems when they will take a job position and will strive to implement and to handle software applications.

Junior professionals normally fill the gap by training on the job (1) or by the assistance of a mentor (2) or even by the attendance of a professional course (3). However these three educational approaches normally cover narrow areas of specialization and learners strive for exhaustive preparation.

We are fully aware of the negative consequences coming from simplified lessons on computer science (CS) and of limited training that technicians find in the working environment. We have designed various courses that tend to improve the knowledge of students for successful integration into employment. A course on CS has been amply described in [7]. Special lessons on the conceptual and logical DB design have been detailed in [8]. As third, the present paper deals with relational databases index design [9].

2 Teaching indexes

Briefly we remind that an index is an ordered set of values which serves a single database table, in particular an index points to the rows of the intended database table. An index indicates the position of data inserted in one or more columns of the database table (Figure 1).

Indexes can be created by using one or more columns of a database table, providing the basis for rapid and efficient access to records. In fact an index has an order and improves the speed of data retrieval operations on a database. An index is an object stored apart from the database tables, and the disk space required to store the index must be less than that required by the addressed table.

Dozens of database indexing methods have been devised but the majority of modern writers tend to filter this broad matter. Authors show inclination toward a restricted set of solutions. For instance [10] reports three basic methods: single column indexes, composite indexes and a unique indexes; [11] describes b-tree indexes, bitmap indexes, hash table indexes, clustered and non-clustered indexes; [12] examines the methods just listed and adds up composite
indexes, covering sparse indexes and dense indexes.

A second group of writers – usually highly specialized writers – tend to dedicate a manual to a sole index technique so that the indexing contents spread out in several handbooks. Both simplified explanations and illustrations scattered over a considerable number of handbooks cannot help specialists who are called to design and optimize large databases.

We have conducted a two stage research in order to assist technicians who are presently working in the DB domain or will work in the next future. Firstly we gathered the various index formats treated in the large set of technical manuals. In particular we examined the manuals published by the major producers of DB systems (IBM, Oracle etc). Secondly we systematized the collected material and prepared a text-book for educational purposes [13]. We arranged a course whose lessons follow the same order of the book units (Table 1). Hereafter the terms 'unit' and 'lesson' are synonymous.

<table>
<thead>
<tr>
<th>Unit 1</th>
<th>Generalities.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit 2</td>
<td>Different Index Formats and Reasons for Using Indexes.</td>
</tr>
<tr>
<td>Unit 3</td>
<td>Index Enhancements in DB2 v9.</td>
</tr>
<tr>
<td>Unit 4</td>
<td>Indexes for Joins.</td>
</tr>
<tr>
<td>Unit 5</td>
<td>Index Versioning.</td>
</tr>
<tr>
<td>Unit 6</td>
<td>Less-Known Index Features.</td>
</tr>
<tr>
<td>Unit 7</td>
<td>Operational Utilities for Indexes.</td>
</tr>
<tr>
<td>Unit 8</td>
<td>Index Cost.</td>
</tr>
<tr>
<td>Unit 9</td>
<td>Indexes for Star Schema.</td>
</tr>
</tbody>
</table>

Table 1 – Table of Contents

3 Comments on Lessons

After the introduction, the second lesson presents different index formats. Lessons from 3 to 9 expand on various technical topics dealing with indexing techniques. The course explains twenty-one kinds (or formats) of indexes which exhaust current technologies at the best of our knowledge. The Appendix exhibits all the index formats.

Actually an index is defined by using the SQL statement Create Index, and a parameter of this statement frequently equals to the index format label.

We have cataloged the index formats into ten groups (see Appendix) so to help a student understand similarities and differences and to reduce the memory load. Each group of indexes has specific technical features briefly commented aside. A group contains one or more formats; the index formats belonging to a group can be mutually exclusive (e.g. Padded and Not-Padded) or may overlap in a way (e.g. Clustering and Clustered). Students learn to combine the various kinds of indexes so that one can create - for example - an index which is simultaneously Unique, Clustered, Multiple-Columns, Not-Padded, Secondary, Not-Partitioned. Lessons from 3 to 7 dissect pros and cons deriving from each method; this discussion is crucial for an attendee who aims at grasping the possibilities and the limits of a practical solution.

We teach the students to determine the right indexes soon after they will create a database table and to check the designed indexes against various SQL queries for that table. This is because an index can be used for many different reasons and may be influenced by an assortment of functions. In broad strokes an index can be used for:

- Enforcing uniqueness of primary keys and unique constraints.
- Improving performance in data access, through different index probing, such as Index-Only, index Matching, index Screening.
- Avoiding Sorts.

In principle the access to data is faster with an index than with browsing all the table data. For example, to speed up data retrieval against a predicate such as WHERE Capital = 'Paris' one can create an index on the ‘Capital’ column of the ‘Nations’ table to easily locate a specific capital and avoid reading through each row of the table 'Nations' (or scanning the whole table). An index can improve performances in several ways which we comment on during the course and briefly sum up as follows:

1) Indexes can be used to reduce the number of get pages and input/output operations.
2) Applying an SQL predicate to the index may be more efficient than applying the same predicate to the data. The use of an index can reduce the number of data pages and rows scanned. If a row does not qualify the predicates in the index, there is no need to retrieve the data page. This applies to SQL statements that access only a few rows of data in a table, but also to processes that sequentially access a large quantity of data in the table, and to single table access, as well as multiple table access.
3) Reducing the number of data pages accessed is particularly interesting if the access is via an index that has a low cluster ratio (i.e. a low order of data row). A low cluster ratio could increase the probability of synchronous input/output operations and therefore the cost of the statement.
4) The best option is to make the predicate indexable. However, if this is not possible, one can still benefit from index screening or index-on expression.

5) Using an index can completely eliminate access to data pages when index-only access is possible.

6) Because data access is completely eliminated when using index-only access, such an index creates an ‘alternative’ clustering sequence for those queries.

7) An index can also eliminate a Sort (due to Order By, Group By, Distinct, Join) if the requested order matches with the index columns. A sort can be eliminated by scanning an index, no matter the direction (forward index scan or backward index scan). Please note that any index can avoid Sorts, not only clustered indexes.

Indexes have a cost of maintenance and the lessons address the students to evaluate the access speed improvement that an index provides against the cost of maintaining that index. Listeners are carefully invited to consider the price of each technical improvement. We summarize a number of reasons explained in the last lesson of the course.

a - Indexes require storage space. Padded indexes require more space than not padded indexes for long index keys. For short index keys, not padded indexes can take more space.

b - Each index requires an index space and a data set, or as many data sets as the number of data partitions if the index is partitioned, and operating system restrictions exist on the number of open data sets.

c - Indexes must be changed to reflect every insert, update or delete operation on the base table. If an update operation modifies a column placed in the index, then also the index must be changed. The time required by these operations increases accordingly to the number of indexes.

d - Indexes can be built automatically when loading data, but this takes time. They must be recovered or rebuilt if the underlying table space is recovered. These operations might be time-consuming.

Appropriate indexes speed up the data retrieval considerably, but in counterpart the operations which need to maintain the index, such as Update/Insert/Delete statement, require extra work when an index is involved. In fact not only the intended row has to be updated/inserted/deleted into the table, but it also has to be updated/inserted/removed in all the indexes that are defined on that table. The lesson concludes that the cost of maintaining multiple column indexes can sometimes be higher than the costs of maintaining the data.

4 Didactical Conclusions

The present educational project produced an advanced course on DB design, and eleven editions of this course have been held in the United States, Germany and Italy on request of organizations and individuals. Globally one hundred students attended the lessons; all were male. The attendees can be subdivided into junior technicians (18-22 year old) and practitioners at various levels of specialization in the database domain; the latter made the majority group (over 95%).

Lessons have been illustrated through oral teaching and labs which globally last eighteen hours and cover three working days. The didactical material – the text-book and the slides – have been arranged in the following manner. Every chapter of the book encompasses a fixed number of topics and a slide summarizes each topic. A topic begins with a new page in the book that exhibits the relative slide on top. The subsequent pages illustrate the intended topic in any detail and are verbally commented by the teacher. This accurate correspondence amongst a projected image, the pages of the book and the oral lessons has been appreciated by the attendees.

We do not have formally evaluated this course except for the satisfaction factor (SF) whose mean lies over 98 points in eleven editions. A student was allowed to express his satisfaction using five values: 100 (very-satisfied), 75, 50, 25 and 0 (absolutely-unsatisfied). In addition students manifested their oral feedback on the good quality of the material and the educational method which was easy to follow. Experienced participants provided the most significant return of information about this course as they used the lesson contents to solve practical issues they faced in the living environment in advance of the lessons. The answers to unresolved problems emerged as the most immediate and satisfactory fall out of this course.

These researches aimed at out-pacing current restrict visions on the indexing techniques by providing the systematic account of the index formats. This was the main didactical purpose of the present work. We summed up numerous manuals which turn out to be impracticable or too expensive for a practitioner.

The personal working experience of the authors in the DB domain suggested how to improve the didactical material which can be exploited in a variety of educational environments. For example a teacher can reshape and ameliorate a graduate course on DB using the Appendix. This easy summary makes an expert-to-be aware of all the methods in use to optimize a database design.
5 Appendix

<table>
<thead>
<tr>
<th>Group</th>
<th>Index Format</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Single-Column, Multiple-Column.</td>
<td>An index can be created on one column only or on many columns simultaneously (max 64).</td>
</tr>
<tr>
<td>2</td>
<td>Unique, Unique Where Not Null, Duplicate.</td>
<td>Unique indexes ensure that a value never repeats in the table columns. For example, the 'Nations' table does not allow duplicate 'Capital' attributes. Creating a unique index prevents duplicate values. This restriction can be applied to generic values (including Nulls) or otherwise to values that are Not Null. Duplicate indexes allow repetition of the same index-key value into the table.</td>
</tr>
<tr>
<td>3</td>
<td>Clustering, Clustered, Not-Cluster.</td>
<td>A cluster index suggests the physical order in which you would like the rows to be stored in the data table. Since rows can be stored only in one physical sequence, a practitioner can define at most one clustering index per table. Both unique and duplicate indexes can be defined as cluster. When you just create an index with the 'Cluster' option, that index is called Clustering index. When the table rows are perfectly in order according to this index, the index is also a Clustered index.</td>
</tr>
<tr>
<td>4</td>
<td>Padded, Not-Padded.</td>
<td>This feature does not pertain to indexes with fixed length columns. These clauses suggest how varying-length string columns are to be stored in the index. A padded index has varying-length string columns filled with blanks characters to their maximum length. In a not-padded index the varying length string columns are not padded to their maximum length in the index. Instead the length information for a varying-length column is stored with the key.</td>
</tr>
<tr>
<td>5</td>
<td>Ascending, Descending, Random.</td>
<td>This is the direction of the keys sequence inside the index, as any index is always an ordered set of values even when the option Random is specified.</td>
</tr>
<tr>
<td>6</td>
<td>Partitioning, Partitioned, Not-Partitioned.</td>
<td>An index is partitioned when the index itself is subdivided according to the partitioning scheme of the underlying data. An index is considered a partitioning index if its leftmost key columns match the columns specified in the partitioning key of the table. The leftmost index key columns are in the same sequence and have the same ascending or descending attributes of the partitioning key of the table.</td>
</tr>
<tr>
<td>7</td>
<td>Primary, Secondary.</td>
<td>A primary index is created for the primary key. Any index on a partitioned table space whose leftmost key columns do not coincide with the partitioning columns of the table is a secondary index. A secondary index can be partitioned or not. A partitioned secondary index is spread on different physical data sets, one for each index partition. Its leftmost columns are different from the columns which partition the table. A non-partitioned secondary index is not spread on different partitions and its columns are different from the partitioning columns of the table.</td>
</tr>
<tr>
<td>8</td>
<td>Indexes on Expressions</td>
<td>An index on expression is created on column expressions and not only just on simple columns. Indexes on expression can be used to retrieve data whose predicates are expressed by using a general expression. To exemplify, these indexes are good for solving predicates such as &quot;where col1+col2 &gt; 100&quot;. In contrast to simple indexes whose key values are the ones contained on simple table columns, the key values of an index on expressions are not exactly the same as values in the table columns. Instead, these index key values are the result of the specified expressions. Also an index on expression can have at most 64 columns.</td>
</tr>
<tr>
<td>9</td>
<td>B-tree</td>
<td>The structure of an index is a balanced tree (B-tree), automatically maintained by the database management system. At the bottom of the tree are the leaf pages of the index. Each leaf page contains a number of index entries consisting of the index key itself and the pointers, known as 'record identifiers', which one uses to locate the indexed data rows. Each entry in the intermediate nonleaf index page identifies the highest key of a dependent leaf page along with a pointer to the leaf page's location. At the top of the tree, a single page, called 'root page', provides the initial entry point into the index tree structure.</td>
</tr>
<tr>
<td>10</td>
<td>Bitmap</td>
<td>These indexes are appropriate for columns which have a small number of distinct values, e.g., male and female. A bitmap index is smaller than a B-tree index and stores only the row-id and a series of bits. In a bitmap index, if a bit is set, it means that a row in the corresponding row-id contains a key value. Also Bitmap indexes can be very helpful in data warehousing because in the depicted situation they can be quite fast when you are only selecting data.</td>
</tr>
</tbody>
</table>
References


