International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 05 | May 2021 www.irjet.net p-ISSN: 2395-0072

Indexing Strategies for Performance Optimization of Relational Databases

Praveena M.V.[1], Dr. Ajeet A. Chikkamannur[2]

[1]*Assistant Professor, Department of Computer Science & Engineering, Dr Ambedkar Institute of Technology, Bangalore, VTU, Karnataka, India*
[2]*Professor, Department of Computer Science & Engineering, R L Jalappa Institute of Technology, Doddaballapur, Bangalore, VTU, Karnataka, India*

Abstract: *Databases and database management systems have been the backbone of computing world for the past many years. The enterprise, web and cloud computing market is growing bigger in terms of size. It will definitely continue to gain prominence in the coming years. With the standardization and consolidation of information technology systems in most enterprises, the demand for highly scalable, reliable and faster relational database systems is on the rise. The databases are crucial for any enterprise operations and to ensure the operations go on smoothly without any issues, database performance is highly crucial. The high performance of the databases could be very well managed by practicing and adopting good database optimization strategies. Indexing is one of the most important strategy to assure the optimal performance of relational databases. To fix the problem of poor database performance and improve the database performance optimization, indexing strategies are essential. Index is basically a data structure based on one or more columns of the database. With faster data retrieval and minimal disk accesses for each query, indexing strategies emerge as powerful technique for performance optimization of relational databases.*

Key Words: **Database, Enterprise, Performance, Indexing, Query, Optimization**

1. INTRODUCTION

In our everyday life in the modern society, people come across databases and database systems quite frequently. In current highly competitive marketplace, information is a strategic weapon. Databases play an important role in almost all areas where computers and mobiles are used. Every day, the demand for good databases and database management systems is exponentially increasing. Enterprises need skilled database professionals to design and manage their databases. The employees need to know the database basics to analyse and use it for effective decision making. Hence, the demand for professionals with good database knowledge is increasing and will continue to do so in future.

In the traditional database applications such as banking, inventory, super market and reservation systems, most of the information stored and retrieved is either text or numeric in nature. In advanced database applications such as multimedia databases, geographical information systems, warehouse, mobile, web and spatial databases, the information is mostly in the form of pictures, videos and wave files. Hence, designing an appropriate access plan and access strategy can be one of the more troubling aspects of developing an efficient relational database applications. In order to assure optimal performance when accessing data in a relational database is to create the correct indexes for tables based on the user queries.

Database design properly forms part of the broader process of systems analysis and design. The Structured Query Language (SQL) is one of the most important elements of modern database technology. SQL is used extensively by virtually all database systems and acts as a major vehicle in facilitating inter database communication. It is a comprehensive relational database language, used to define and manipulate databases.

Indexing is used speed up the retrieval of data from the databases. Indexes are an implementation requirements of practical systems rather than a mere theoretical feature. Availability of very efficient indexing system accounts for the success of the relational databases. Major benefits of indexing are faster access to individual rows and accessing rows in a prescribed sequence.

Indexing a table is an access strategy that is a way to sort and search in the table. Indexes are important to improve the speed with which records in database table can be located and retrieved. Basically an Index is an ordered list of the contents of a column or a group of columns of a table. Commonly used commercial database tools are

generally based on a methodology that enables tables indexing for independent SQL queries [1]. Indexes are used in context like a phone contact list, where the contact details may be physically stored in the order one add people's contact information, but it is easier to find contact when listed out in some alphabetical order.

1.1 Data Structures for Indexing

An index is basically a data structure that holds the values for a certain specific column of a table. Index helps us avoid a full table scan while retrieving data from a table.

- B-trees are the most commonly used data structures for indexes. A B-tree is a balanced tree, yields a uniform search speed regardless of the value of the search key. Insertion, deletion and search operations are done using B-trees in logarithmic time. B+ tree and B* tree are the variants of B-trees.
- Hash tables referred as hash index used in queries that looks for agile execution with exact matches. Here key is the column value and the value in the hash table is a pointer to the row in the table.
- R-trees are tree data structures commonly used with spatial databases. It helps to answer queries such as "find all the medical shops within 3 kilometres of my current location". R-tree main idea is to group nearby objects and represent with minimum bounding rectangles. Hence the "R" in R-tree refer to Rectangle. Some of the variants of R-trees are Priority R-tree, R+ Tree, R* Tree and Hilbert R-tree.
- Bitmap index works for low-cardinality columns that have many instances of values with low selectivity. A column with Boolean values can use Bitmap index.

1.2 Specifying Indexes

Consider the following SQL query statement:
SELECT empname, salary
FROM employee
WHERE empno = '00146' AND deptno = 'CS';
Different types of indexes can be specified on the above query. The possible indexes that could be created may probably looks like this: Index1 on empno, Index2 on deptno and Index3 on empno and deptno. This is a good strategy, and Index3 is probably the best among the three indexes specified. It enables the database system to use the index to immediately look up the row or rows that satisfy the two simple predicates in the WHERE clause. Likewise varieties of indexes such as simple, composite, unique, reverse, bitmap and compressed can created on a database table.

2. RESEARCH APPROACH

A systematic literature review of Indexing Strategies for Performance Optimization of Databases was conducted. An Index in a databases works in the same way as the library system catalog. A library will be having its books cataloged in many ways such as author, course, title, and so on. When one wants to look for book by an author, has to choose the author catalog or the catalog where the books are sorted using author names. The catalog will be having the book information and helps to locate the book among of thousands of books in the library setup. The problem with the catalog appears when the number of books and the size of the catalog grows. Managing multiple catalogs in a library will become a tedious task. In the real world commercial databases with the large number of records, indexes with the book catalog will be too large to handle efficiently. Hence the more sophisticated indexing strategies for databases are necessary in order to optimize database performance.

A database index works similar to a library index. Using appropriate indexes in a large and complex database tables is the most important aspect of database optimization. In the Comparative Study of Indexing Techniques in DBMS [2], the authors claims that the indexing provides random lookups and efficient access of ordered records in a database. Microsoft SQL Server uses non-clustered indexes by default and Oracle database uses B-Tree Indexing technique. Where ever selection queries are frequent, then use Indexing for retrieval of data. While testing relational database query optimization strategies [3], the authors abstracted a language like SQL provides the designer with a tool for specifying what the needed results of the query are without the need to specify how these results should be obtained. SQL Indexes exist to help the faster execution of queries. Indexing is used to reduce the computational and input output subsystem loads when retrieving data [4]. The ubiquity of relational database management systems produced a comprehensive set of strategies to optimize the performance of database queries. Using indexes in databases is the main task to improve query performance, since the indexes are the most used strategies to accelerate query response [5]. The main purpose of using an index is to optimize speed and performance in finding relevant results for a search query. Without using an index, the search engine might scan every data in the database, which consumes considerable time and computing power. For example, using indexing, 10,000 records can be queried within milliseconds,

whereas sequential scan of every record might take minutes. Antonin Guttman [6] proposed dynamic index structure called R-trees to handle spatial data with multi-dimensional spaces. An R-tree is a height-balanced tree similar to a B-tree, with index records in its leaf nodes consists of pointers to data objects. The R-tree index is completely dynamic in nature, and used to manipulate spatial data.

3. INDEXING STRATEGIES

Indexing involves forming a two dimensional matrix completely independent of the database table on which the index has been created. Index consists of Search Key and a Data Reference columns. One column holds the sorted data extracted from the table column upon which the index is created. Another column called the address field identifies the respective disk block where the particular key value can be found.

3.1 Clustered Indexes

Clustered indexes are the unique index per table and uses the primary key to organize the data available within the table. Clustered indexes can greatly increase overall speed of retrieval, but usually only where the data is accessed sequentially in the same or reverse order of the clustered index, or when a range of items is selected. It is very important to have a clustered index on every table. The clustered index is automatically created when the primary key is defined:
CREATE TABLE Customers (custid INT PRIMARY KEY, custname VARCHAR2, city VARCHAR2);
The created table, "Customers", will have a clustered index automatically created, organized around the Primary Key "custid" called "Customers_pkey".

3.2 Non-Clustered Indexes

The non-clustered index contains the index keys in sorted order, leaf level of the index contains the pointer to the page and the row number in the data page. It is typically created on column used in JOIN, WHERE, and ORDER BY clauses. These indexes are good for tables whose values need frequent modification. Microsoft SQL Server creates non-clustered indexes by default when CREATE INDEX command is given. To create an index to sort customer names alphabetically:
CREATE INDEX Customers_custname_asc ON Customers (custname ASC);
This will create a non-clustered index called "Customers_custname_asc", indicating that this index is storing the customer names from "Customers" stored alphabetically in ascending order.

3.3 Covering Indexes

Covering Indexes contains all required information to resolve the query. It includes all the columns, the query refers to in the SELECT, JOIN, and WHERE clauses. The performance benefit gained by using covering indexes is typically great for database queries that return a large number of rows.

3.4 Filtered Indexes

Filtered index is an optimized non-clustered index especially suited to cover database queries that select from a well-defined subset of data. It uses a filter predicate to index a portion of rows in the database table.

3.5 Index Selectivity and Index Density

Index selectivity represents the number of distinct key values in the database table. The perfectly selective keys are UNIQUE KEY and PRIMARY KEY. Index Density represents number of duplicate key values in the database table. Hence, more selective indexes will have lower density and the best indexes will be ones with highest selectivity.

3.6 Best practices for creating indexes

- Create the clustered index on every table in the database.
- Keep indexes lean with one or few columns.
- Create the clustered index on the column with high selectivity.
- Try to eliminate duplicate indexes such as multiple indexes created on the same set of columns in a table.
- Create Non-clustered indexes in different file groups which can reside on separate disk drives to improve the data access.

3.7 Improving indexing strategies

Indexing is too often overlooked during the design and development process.

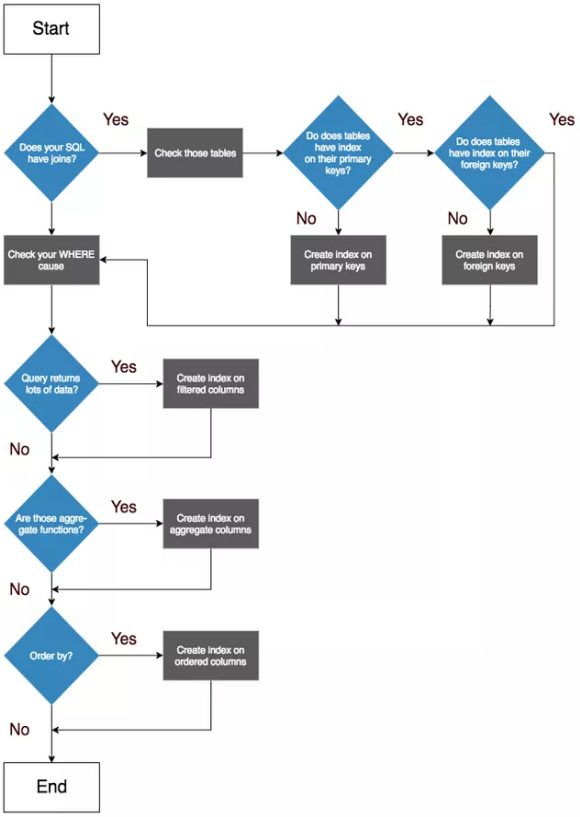

Figure 1: Structuring queries with indexing strategies

A good indexing strategy is the best tool for tuning and optimizing the database performance. Adopting indexes in queries usually does not require much coding, but it does take a bit of thought and creativity. Figure 1 outlines the best practices for structuring queries with indexing strategies during database development process.

Indexes should be built to optimize the access of database queries. The optimal set of index creation requires a list of the SQL statement to be used, an estimate of the frequency that each statement will be executed, and the importance of each query. Only then the task of creating the right indexes to optimize the right queries can be achieved. Sometimes there is an advantage to include additional columns in an index to increase the chances of index-only access. There is no arbitrary limit on the number if indexes specified for a particular database table. Hence creating and adopting indexing strategies is necessary to support the database queries.

4. CONCLUSIONS

Indexing strategies could be very fruitful in optimizing the database when used carefully. A well indexed database strategy can be very efficient and high performing with a quick turnaround time of data access. These strategies could play an important role in making the database operations seamless with quick database access. Hence these proposed indexing strategies are helpful in performance optimization of relational databases.

Indexing is very helpful in database optimization as it minimizes the number of data access for processing a database query. Indexing strategies could drastically improve the query execution plan and the query runtime.

REFERENCES

1. Radoslaw Boronski and Grzegorz Bocewicz, “Relational Database Index Selection Algorithm ”, Springer International Publishing Switzerland 2014, CCIS 431, pp. 338–347, 2014.
2. Manoj Kumar Gupta and Dr. Dharmendra Badal, “Comparative Study of Indexing Techniques in DBMS “, Business Intelligence and Data Warehousing Conference, USMS, Delhi, India, March 2012.
3. Vlad Diaconita, Ion Lungu, Iuliana Botha ,“Testing relational database query optimization strategies”, ITCN 12, Vienna, Austria, pp. 152 – 157, November 2012.
4. Derek Colley, Dr Clare Stanier, “Identifying New Directions in Database Performance Tuning”, Elsevier, Procedia Computer Science 121 (2017), pp. 260–265, 2017.
5. Alberto Arteta Albert, Nuria Gómez Blas, Luis Fernando de Mingo López, “Intelligent Indexing—Boosting Performance in Database Applications by Recognizing Index Patterns”, Electronics 2020, MDPI, Basel, Switzerland, 20 August 2020.
6. Antonin Guttman, “R-Trees: A Dynamic Index Structure for Spatial Searching”, SIGMOD 1984.
7. Ramez Elmasri, Shamkant B. Navathe, Fundamentals of database systems, 6th ed, Addison-Wesley Publishing House, 2011, ISBN 13: 978-0-136-08620-8.
8. Yannis E. Ioannidis, Query Optimization, Computer Sciences Department, University of Wisconsin.
9. Jonathan Lewis, Cost-Based Oracle Fundamentals, Apress, ISBN (pbk): 1-59059-636- 6, 2006.
10. Kimberly Floss, Understanding Optimization, Oracle Magazine, 2005.

11. Oracle® Database Administrator's Guide 11g Release 1 (11.1).

12. Michelle Malcher, Oracle Database Administration for Microsoft® SQL Server® DBAs, The McGraw-Hill Companies, ISBN: 978-0-07- 174430-0, 2011.