Physical Database Design Techniques to Improve Database Performance

Syed Ateeq Ahmed

---***---

Abstract - *Performance tuning of a database is one of the crucial responsibilities of a Database Administrator (DBA). There are a vast number of areas where the DBA need to tune in order to improve performance of the database. However, improving performance on the go can be a challenging task for any DBA. In this research paper the main focus is on improving the performance of database by applying correct physical database design techniques. Applying valid techniques can improve the performance of the database drastically. In this research paper, key techniques to improve the performance have been identified and discussed.*

Key Words: B-Tree, Bitmap, OLTP, DSS

1. INTRODUCTION

Physical database design is to maximize the efficiency of data processing. When designing a database, the main focus is on minimize the execution time of queries. The physical database design process requires to take several crucial decisions that will have an impact on the performance of the database applications. These decisions include the storage format, designing fields, appropriate usage of data types, coding style etc. However, the main focus of this research paper is on major physical database design areas which will have performance impact. In this research paper, the focus is on the following major physical design issues:

- Indexing
- Denormalization
- Data Partition

1.1 When to use indexes

As the size of the databases are very large, mostly in terabytes, searching for data is a time consuming process. Full-table scans should be avoided. To access the data block quickly, indexes can be used. Indexes provide pointers to the data block where the intended data is available. For example, an index can be created to find the employee details given a particular Employee ID. The Database Administrator (DBA) creates indices in a database as a part of performance tuning techniques. However, use of indexes should be done with care, as too many indexes on a table could damper the performance.

The main purpose of physical database design is to enhance the performance of the database application. During the database design, the DBA must identify the attributes to create indexes. In certain situations, there is a possibility that indexes can decrease the performance. DBA must make a tradeoff between the performance gains and the overhead that will occur due to the creation of indexes. The following are the guidelines that the DBA and the database designers must follow to fine tune the database performance:

- Indexes are mostly useful on larger tables
- Index the columns that are in the predicate clause of join conditions. Creating indexes on foreign key columns as they are used mostly in the join conditions.
- Index the columns that are used in ORDER BY and GROUP BY clauses.
- Create an index if you need to access less than 15 per cent of table data.
- Columns data having large number of characters is poor choice for indexing
- Avoid indexing on the columns which are frequently updated.
- Choose the columns to index which have high selectivity.
- Avoid creating too many indexes on a table.
- Avoid index on columns that have too many null values.

Finally, the golden rule is that we should consider the type of queries that are expected to occur as against the table's columns.

1.2 Type of Applications

Most of the applications developed are of the type online transaction processing (OLTP). Examples of OLTP systems are order entry, sales and purchase processing, airline reservation etc. In an online transaction processing system, transactions are short, however, large number of users shall be online and operations on the database are very frequent. Operations on the database in OLTP applications can be data manipulation or retrieving information from the database.

In contrast, data warehouse applications contain very large databases, mostly historical data. In case of decision support system, few number users will be online and large volumes of data are used in data processing and performing analytical operations on the database.

The types of indexes are crucial and the database administrator must take care in choosing the appropriate index type. In case of an incorrect index type, the performance of the database application can be adversely affected.

1.3 Type of Indexes

Due to the different types of applications, the type of data stored in the database, format of data, and most importantly the type of queries that run against the database are some of the reasons for choosing appropriate type of index. The database administrator must be well averse with the types of indexes in order to make appropriate decision in choosing the relevant index type. The following are some of the types of indexes. The database administrator has to choose the relevant index based on the type of information that needs to be retrieved from the database.

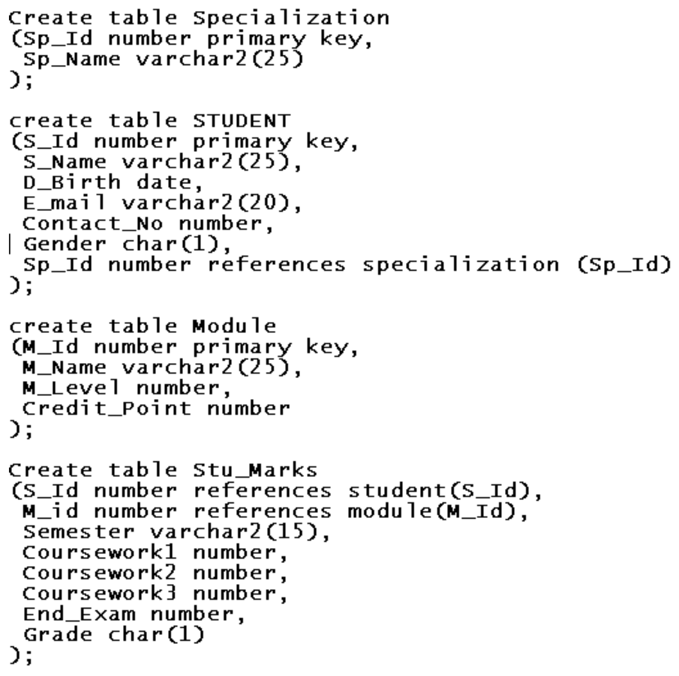
- Unique index
- Concatenated index
- B-tree Index
- Bitmap index
- Function based index
- Partitioned index
- Reverse key index

2. Denormalization

Database normalization is done to remove the redundancy of the data so as to maintain the integrity of the database and to eliminate the redundant data. Normalizing a database is part of an effective database design process whereby a table is decomposed into two or more tables with an intention to remove anomalies. A normalized database can be avoid various anomalies, however, may have a negative impact on the performance of the database. The execution time of queries on normalized database is considerably high due to the multiple joins across the tables. Normalized tables can lead to inefficient data processing.

To improve the performance of the database, denormalization of a database can be purposely done where tables are transformed into non-normalized form. This can be achieved by adding redundant columns to tables. Derived columns can also be added to denormalize a database.

Consider the following table structures:



```
Create table Specialization
(Sp_Id number primary key,
 Sp_Name varchar2(25)
);

create table STUDENT
(S_Id number primary key,
 S_Name varchar2(25),
 D_Birth date,
 E_mail varchar2(20),
 Contact_No number,
|Gender char(1),
 Sp_Id number references specialization (Sp_Id)
);

create table Module
(M_Id number primary key,
 M_Name varchar2(25),
 M_Level number,
 Credit_Point number
);

Create table Stu_Marks
(S_Id number references student(S_Id),
 M_id number references module(M_Id),
 Semester varchar2(15),
 Coursework1 number,
 Coursework2 number,
 Coursework3 number,
 End_Exam number,
 Grade char(1)
);
```

Fig-1: Sample table structures

If we need to generate a student report to display the marks obtained by a student along with name of module, name of specialization and name of the student, the query will take considerable time to execute as it involves multiple join conditions. Performance gains can be achieved by including the relevant redundant columns to the Stu_Marks table. Another way to improve performance would be to include the derived attributes in the table Stu_Marks e.g. total marks.

3. Partitioning

Queries that run against large volumes of data could be time consuming and will have negative impact on the performance of the database. Consider the student marks table given in figure-1. As the students' data gets accumulated over the years, the Stu_Marks table can have millions of records within a few years' of time span. If we need to run queries against such large tables, data partitioning can be an effective mechanism to achieve performance gains.

Depending upon the requirement, partition can be done as either a vertical partition or a horizontal partition. Consider the Stu_Marks table shown in Figure-1, if we need to work on the current students data, we can perform a horizontal partition separating the historical data with the current data. Students who have already completed the course can be segregated to separate table. If the data volume increases many folds, the Stu_Marks table can be split up further based upon the Specialization of the student. In case of a table having a very large number of columns, vertical partition can be done by segregating columns of our choice into separate tables. Partitioning of data brings further benefits in the form of ease of managing the data and availability of the data.

Data Partitioning can be a very effective mechanism which enables the database administrators and database designers to easily handle the multi-terabyte systems where the availability of the data is a key requirement. Partitions are also useful in parallel processing environments where server1 can work on partition1, server2 can work on partition2 etc. increasing the performance of the database.

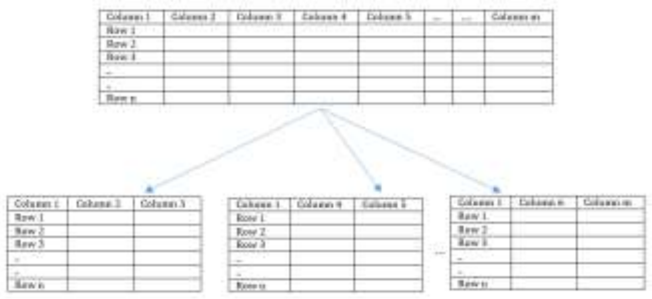


Fig-2: Vertical Partition

The figure-3 shows how a horizontal partitioning can be done on a table data. In case of horizontal partitioning, number of columns shall remain the same in the partitioned tables. It can be done to avoid long full table scans.

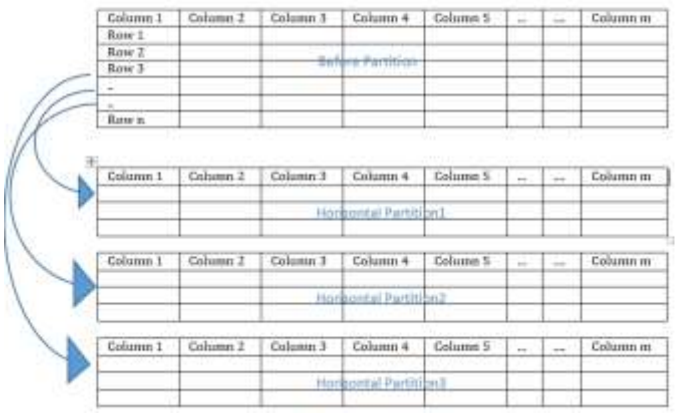


Fig-3: Horizontal Partition

4. CONCLUSIONS

In this research paper, crucial aspects of physical database design having an impact on performance have been discussed. There are certain other physical design areas where the Database Administrator and Database Designers need to focus to achieve performance gains.

REFERENCES

[1] Patil, S., Damare, P., Sonawane, J., Maitre,N., "Study of Performance Tuning Techniques," JETIR, 2015.

[2] Matalqa, S.H., Mustafa, S.H., "The Effect of Horizontal Database Table Partitioning on Query Performance", IAJIT, 2016.

[3] Hoffer, J.A., Ramesh, V., Topi, H., "Modern Database Management System" , 10th edition, 2011.

[4] Baer H., "Partitioning in Oracle Database 11g", Oracle, 2007.

[5] Agrawal S., Narasayya, V. and Yang, B., "Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design", SIGMOD, 2004.