DOI: https://doi.org/10.33216/1998-7927-2019-253-5-78-83

UDC: 004.657

DATABASE QUERY OPTIMIZATION

Nesterov M.V., Bakitko D.E., Mikhaylova A.O.

ОПТИМІЗАЦІЯ ЗАПИТІВ БАЗИ ДАНИХ

Нестеров М.В., Бакитько Д.Е., Михайлова А.О.

This article reviewed the methods directed at optimizing the database. The goal was to find the most suitable method for the quick execution of queries.
Keywords: *database, indexing, clusters, optimization, partitioning, statistics, SQL.*

Introduction. The processing time of the request also determines the speed of the database, which characterized by amount of time per request processing. The execution time of a request may also depend on complexity of the request. As a rule, the request could be substantially simplified or you can do several simple requests that will do the same job as a complicated one, but much faster.

Formulation of the problem. If a web service loads for a very long time, then this causes a delay in the business processes of the application and service. By optimizing database queries, you can increase the speed of the application. The request processing time is a very important criterion for evaluating database performance. When designing a system, it is important to predict an increase in database queries, as well as an increase in data volumes. The increase in performance consists of the execution time of queries, the speed of information retrieval in non-indexed fields, the greatest number of parallel access to data. MySQL database will be used as an object of research in the article.

Methods of setting productivity. Methods such as indexing, query code optimization, clustering, and partitioning will be considered.

A. INDEXING. The index is an acceleration of the search operation for records in the table and performance of other operations that relate to the search: sorting, retrieving, modifying. All information about indexes stored in index files, which consist of data and indexes of record numbers. The index field requires data from the index file, and the pointer serves to link the index file entries. To create an index, it is necessary to specify the field of the table to which it is necessary to assign indexing.

The created indexes could be used in such ways as:

1) Sequential data access in a given sequence of values of the index field.

2) Direct access to individual records of the index file in a given sequence of the index field.

Information storage with indexing involves the use of two stored files.

Indexes are used to speed up data retrieval, which retrieved faster by reducing the number of disk I/O operations for which pointers are used.

Not necessarily every entry in the indexed file must contain an index. This index called loose. Loose indexes are small, due to which browse the contents of the database.

1) The file of the table in which data stored, such as, information about users of the resource.

2) File with indices of blogs that users have created on the resource.

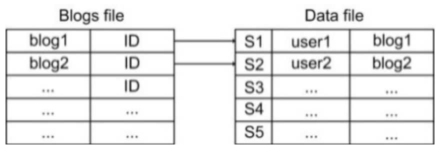


Fig. 1. Index data storage structure [1]

The file with indexes of blogs will be arranged alphabetically, and the file with data depending on the content. If you need to organize by personal information, you need a file that stores it.

Indexing is very effective if the number of records sought does not exceed 10-15%. If the proportion of the files you are looking for is too large, the index file will be used too often, which will lead to a losing strategy for using it. In these cases, it is necessary to execute a direct search, without using indexing [2].

All found data follow each other, which means that their position is very quickly determinate and read. Identifiers of the fields found a point to the required records [3].

Each index must be unique. Unique indexes exclude the same data. Creating uniquely makes sense only if the data itself could unique. In the case of duplication of data, there will be an error and it will not be possible to save the entered information.

Unique indexes have such advantages as [4]:

- Integrity of the data in the columns of the tables.
- Provide more information.

There are such types of indices [5]:

- Bitmap indexes.
- With reversed key.
- With a compressed key.
- Based on features.
- Partitioned.
- Global.
- Local.
- Invisible.

To create indexes in Oracle, you must adhere to the recommendations:

- Do not create indexes for small tables.
- It is necessary to create primary keys for all tables. In the case of the name of the primary key will be automatically created an index on the primary key.
- It is necessary to index the columns of the tables that take part in table connections.
- Indexing will be very efficient for using the column by the WHERE clause.
- Columns that updated should not be indexed.

For each table, the index must take into account the operations that will be performed on the columns. Creating an index in Oracle:

```
CREATE INDEX blog_id ON blog(blog_id)
    TABLESPACE INDEX_BLOG;
```

When creating indexes, you should follow the recommendations:

- Do not create indexes for columns on which INSERT, UPDATE, and DELETE operations are often performed.
- Do not use indexing for small tables.

Create SQL index:

```
CREATE INDEX idx1 ON blogs (col1);
```

Indexing allows you to not view the entire table for data retrieval, thereby increasing database performance.

B. PARTITIONING. Partitioning meant breaking a large table into less to promote the execution of necessary queries. Partitioning capabilities include partition independence, which makes it possible to carry out backup, restore, and index creation operations specifically on a partition, rather than on the entire large table, which will significantly reduce database idle time [6].

Partitioning improves the performance of information processing in large tables, but it does not protect against poor-quality queries.

The advantages of partitioning [6]:

- Accessing data subsets while maintaining their integrity.
- Fast execution of maintenance operations achieved by working with one or several sections, and not with the entire table.

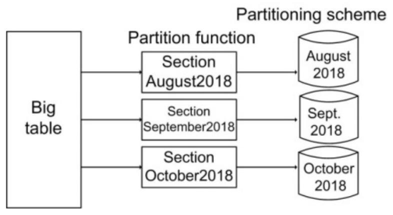


Fig. 2. Partitioning structure [7]

In Oracle Databases, archiving old data that older than the set date is common. With partitioning, this procedure is very simple and effective. This property is typical for large repositories.

There are six ways to partition tables in Oracle:

- Partitioning by key ranges.
- Interval partitioning.
- Hash partitioning.
- According to the list of key values.
- Reference partitioning.
- System partitioning.

Using interval partitioning, it is possible to create sections based on ranges of values:

```
partition by range (sales_dt)
(
partition b0001 values less than (to_date('2019-04-01','yyyy-mm-dd')),
partition p0002 values less than (to_date('2019-05-01','yyyy-mm-dd'))
```

Sections are indicated for April and May 2019.

The table will be divided into sections by date range.

If the data are unevenly distributed over time intervals and it is necessary to know not only the data for April and May but also the data that were before the specified interval, then you should use hash partitioning. To do this, you need to select the number of sections, after which Oracle will define the hash value of the key of each row and put it into the necessary section [8].

```
CREATE TABLE users_data
(ticket_no NUMBER,
user_id INT NOT NULL,
user_name INT NOT NULL,
user_bir INT NOT NULL)
PARTITION BY HASH (user_key)
PARTITIONS 4
* STORE IN (ts1,ts2,ts3,ts4);
```

It is also possible to partition by key, even if the given key does not exist in the table itself. The data method called virtual column partitioning. Early versions of Oracle memorized a non-existent column into a trigger, due to which performance was reduced due to calls to the trigger. The Oracle Database 11g specification makes it possible to create a column without storing it in a table. This column will be calculated during operation [9]. The complexity of the virtual column may depend on the amount of computational data.

C. CLUSTERING. The term "clustering" means a group of servers, which is interconnected and behaves like a single database that is capable of processing incoming requests [10].

To check the status of the nodes, the local and cluster network used. Checks such as LooksAlive and IsAlive performed. The first command executed every 5 seconds, which tries to make sure that no network problems detected. The second command performs a deeper check every 60 seconds. In case the second command detects an error, the check will be performed 5 more times. When an error detected again, the cluster will transfer the group to another node and transfer all network resources to another server. When transferring data, the group will be ready for servicing customer requests [11].

The advantages of using clusters [12]:

- Eliminate long database downtime. Resources could be delivered to another server without losing the connection to the database.
- Small time and effort when replacing a cluster with better performance. To do this, add a new node to the cluster, install the necessary updates and exclude the old cluster from the network.

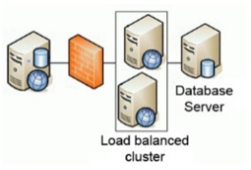


Fig. 3. The structure of the cluster network [12]

There are the following types of building cluster systems [12]:

- With shared disks.
- With shared memory.

Each node of such systems specifically serves its own database fragment. Such systems lack shared memory and storage devices.

When a database uses shared memory and I /O devices, it called Shared everything.

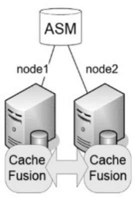


Fig. 4. Shared everything [11]

Such systems characterized by high-speed between nodes and have shared access to storage devices. The strength of the system is fault tolerance, parallel processing, and ability of the network to expand.

A major disadvantage of the system is competition of nodes for input devices and memory. These shortcomings manifest themselves when the network is busy and when executing INSERT, UPDATE, DELETE commands, which need many processor resources to execute.

The Shared Nothing model fixed all the flaws of Shared Everything. Competition disappears due to the lack of shared access of nodes to memory and devices. Each node of the network performs its work separately. Due to this, and increases system performance.

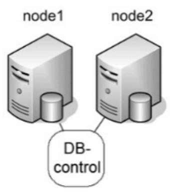


Fig. 5. Shared Nothing [11]

The disadvantage of such an organization is complexity of the network since it is necessary to check the state of each network node, which will entail overhead costs. Backup issues may occur during host status issues. To ensure integrity of the nodes, you must restart the nodes that process execution of the request.

D. Optimization SQL queries. To optimize execution of data change statements, use the EXPLAIN PLAN statement, which allows you to view the execution plan of an SQL statement.

EXPLAIN PLAN allows you to see an execution plan that the analyzer can use to execute an expression [13]. The constructed expression plan is written to the table without saving the SQL expression.

The optimizer's query execution plan runs faster with checking the column index stores than using index row stores [3]. The selection of columns is based on a lower cost value than on rows.

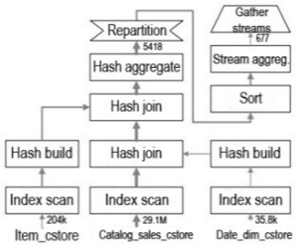


Fig. 6. Optimization using column index repositories [3]

This example demonstrates scanning two tables and building on the basis of hash table scanning. After that, several threads scan the indexes of the column storage indexes and simultaneously check the hash tables. Result sets are collected in one output stream. Such a model is capable of processing more than 144 million tuples of less than a third of a second [3].

Expression plans could be viewed in SQL * Plus using the DBMS_XPLAIN package. This package consists of five functions [14]:

1) DISPLAY - output formatted execution plan.

2) DISPLAY_AWR - output a formatted execution plan from the AWR directory.

3) DISPLAY_CURSOR - output a formatted plan from any loaded cursor.

4) DISPLAY_SQL_PLAN_BASELINE - formatted output of one or several SQL plan expressions by headers.

5) DISPLAY_SQLSET - output execution plan, which stored in the SQL Tuning set.

After generating the execution plan, the PLAN TABLE table is automatically generated, which is a global temporary table that could be used by all users.

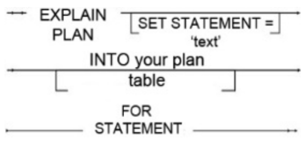


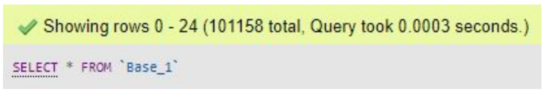
Fig. 7. EXPLAIN PLAN command structure [14]

E. Collecting statistics. During the execution of each SQL query, optimizer looks for the best solution for its execution. Optimizer relies on statistical data, which includes information about distributed data, characteristics of tables and indexes. [15]

For Oracle, automatic database gathering of all optimizer statistics recommended.

Since the release of Oracle Database 10g, a statistics collection tool has been introduced.

Conducting an experiment. A table was created in the database, which consists of 101158 rows.



✓ Showing rows 0 - 24 (101158 total, Query took 0.0003 seconds.)

```
SELECT * FROM `Base_1`
```

Fig. 8. Created table

Based on this table, the methods described will be used. The table consists of columns:

- ID - identifier.
- Users - workers.
- Age - their age.
- City - city of residence.
- Work_with - the date from which the employee got a job.

To create an index, the Work_with column was selected, be able to select the ranges of the dates work of employees:

CREATE INDEX Work_with ON Base_1(Work_with);

To extract information on the date range, the following command executed:

SELECT * FROM `Base_1` WHERE Work_with BETWEEN '1970-04-26' AND '2019-04-26'

The result of the query is an alphabetically sorted list of employees by range.

The index file will look like this:

Table 1

Index file

ID	**Index**
ID user	Date work
…	…

The file stores the user ID and the date of his employment. By identifier, data will be searched from the main table. The main feature of indexing is: there is no need to compare each row of the table with the query condition, just refer to the index file.

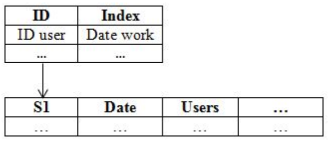
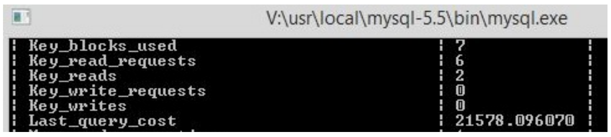


Fig. 9. Index file accesses to the main table

To check the query to the indexed table, a SQL script was created that will give the statistics of the query. SQL script consists of commands:

```
USE database;
FLUSH STATUS;
SELECT * FROM `Base_1` WHERE Work_with
BETWEEN '1920-04-26' AND '2019-04-26';
SHOW SESSION STATUS;
```

This statistic will show the cost of resource consumption when executing a query (Last_query_cost)



V:\usr\local\mysql-5.5\bin\mysql.exe

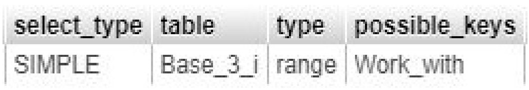
```
| Key_blocks_used                      | 7            |
| Key_read_requests                    | 6            |
| Key_reads                            | 2            |
| Key_write_requests                   | 0            |
| Key_writes                           | 0            |
| Last_query_cost                      | 21578.096070 |
```

Fig. 10. Indexing statistics

The Last_query_cost parameter is 21578.

To get conclusions from the query optimizer, you must enter a query:

EXPLAIN SELECT * FROM `Base_1` WHERE Work_with BETWEEN '1970-04-26' AND '2019-04-26'

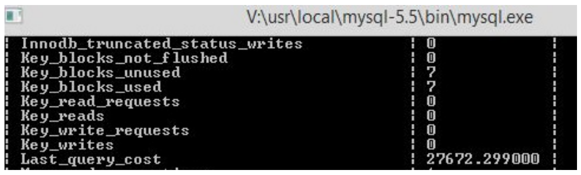


select_type	table	type	possible_keys
SIMPLE	Base_3_i	range	Work_with

Fig. 11. Displays query optimizer

Based on the query optimizer output, it could be concluded that MySQL uses an index for execution of this query, which finds the necessary information in the index file in a certain range (Type: range). 8948 rows were found for this query. There were no failures when executing commands.

To check statistics of execution of a partitioning request, a SQL script launched.



```
| Innodb_truncated_status_writes            | 0            |
| Key_blocks_not_flushed                    | 0            |
| Key_blocks_unused                         | 7            |
| Key_blocks_used                           | 7            |
| Key_read_requests                         | 0            |
| Key_reads                                 | 0            |
| Key_write_requests                        | 0            |
| Key_writes                                | 0            |
| Last_query_cost                           | 27672.299000 |
```

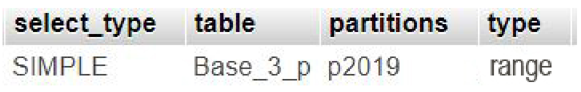
Fig. 12. Partitioning statistics

The Last_query_cost parameter is 27672.3.

The first method consumes fewer system resources for its execution. The fewer system resources consumed, the faster the request executed.

To get conclusions from the query optimizer, you must enter a query:

EXPLAIN PARTITIONS SELECT * FROM `Base_3_p` WHERE Work_with BETWEEN '1970-04-26' AND '2019-04-26'



select_type	table	partitions	type
SIMPLE	Base_3_p	p2019	range

Fig. 13. Query query optimizer output

To execute the query, a call made to partition of the table, which stores the ranges of dates.

As a result, a query for sampling dates taken not from the main table, but from its sections, which means that execution of the query is faster than comparing all the records of the main table with the condition in the query.

Table 2

Query time

	Index	No Index	Partitioning
Query time (sec)	0.0007	0.0055	0.0007

The query indexed and partitioned tables executed in 0.0007 seconds, which is 7.85 times faster than to the usual similar table.

As a result of the analysis of productivity tuning methods, an index method chosen. This method allows you to quickly process a query to the table due to less load on system resources and the ability not to compare each row of the table with the condition in the query, but to directly access the index files. Index files allow you to process a query faster and their implementation in practice is easier than partitioning a table. Indexing should be applied to those rows that are not confirmed by data changes. In the example, this table does not change; it serves to collect company statistics on employees. And also indexing reduces resource consumption when executing a query.

Conclusions. According to the results of the analysis of database productivity methods, it could be concluded that each method has its own advantages and may affect the speed of the query to the database. However, each method implies that to successfully carry out the method, it is necessary to carefully work out the structure of the database in the required subject area. Great emphasis should be placed on the ability to update the columns and operate on their data.

In the analysis performed, the method that most suitable for the database optimization problem was chosen.

References

1. Indexing in databases. https://all4study.ru/bd/
2. Sadhana J. Kamatkar, Ajit Kamble, Amelec Viloria, Lissette Hernández-Fernandez and Ernesto García Cali: Database Performance Tuningand Query Optimization, 2018, 4.
3. Per-Åke Larson, Cipri Clinciu, Eric N. Hanson, Artem Oks, Susan L. Price, Srikumar Rangarajan, Aleksandras Surna, Qingqing Zhou: SQL Server Column Store Indexes, 2011, 2-8.
4. SQL Index Design Guide. https://docs.microsoft.com/ru-ru/sql/2014-toc/sql-server-index-design-guide
5. Oracle indexes. https://oracle-dba.ru/docs/architecture/indexes/
6. Sanjay Agrawal, Vivek Narasayya, Beverly Yang: Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design, 2004, 1-9.
7. Partitioned tables. https://oracle-patches.com/oracle/prof/3006-секционированные-таблицы
8. Eadon, G., Chong, E. I., Shankar, S., Raghavan, A., Srinivasan, J., & Das, S: Supporting table partitioning by reference in oracle, 2008, 2-5.
9. Chakkappen, S., Cruanes, T., Dageville, B., Jiang, L., Shaft, U., Su, H., & Zait, M.: Efficient and scalable statistics gathering for large databases in Oracle 11g, 2008, 1-2.
10. Bertini, L., Leite, J. C. B., & Mosse, D.: Statistical QoS Guarantee and Energy-Efficiency in Web Server Clusters, 2007, 2-3.
11. Cluster database management systems. http://www.jetinfo.ru/stati/klasternye#gl_2_1
12. Database cluster. http://www.r-it.su/solutions/san/nas-db-cluster/
13. Khaled Yagoub, Pete Belknap, Benoit Dageville, Karl Dias, Shantanu Joshi, and Hailing Yu: Oracle's SQL Performance Analyzer, 2008, 3
14. Oracle Database 11g: Performance Tuning - Execution Plans. http://oracleonrussian.blogspot.com/p/oracle-database-11g_10.html
15. Ziauddin, M., Das, D., Su, H., Zhu, Y., & Yagoub, K.: Optimizer plan change management, 2008, 2-5
16. Benoid Dageville, Dinesh Das, Karl Dias, Khaled Yagoub, Mohamed Zait, Mohamed Ziauddin: Automatic SQL Tuning in Oracle 10g, 2004, 7-8.
17. Serge G. Marokhovsky Shuzi Chen, Sadasiva K Prathab, Anthony Ward: System and method for gathering and analyzing database performance statistics, 2002.

Нестеров М.В., Бакитько Д.Е., Михайлова А.О.
Оптимізація запитів бази даних

У цій статті розглянуті методи, спрямовані на оптимізацію бази даних. Мета полягала в тому, щоб знайти найбільш підходящий метод для швидкого виконання запитів.

Ключові слова: *база даних, індексація, кластери, оптимізація, розділення, статистика, SQL.*

Нестеров М.В., Бакитько Д.Э., Михайлова А.А. Оптимизация запросов базы данных

В этой статье рассмотрены методы, направленные на оптимизацию базы данных. Цель состояла в том, чтобы найти наиболее подходящий метод для быстрого выполнения запросов.

Ключевые слова: *база данных, индексация, кластеры, оптимизация, разбиение, статистика, SQL.*

Нестеров Максим Володимирович – старший викладач кафедри комп'ютерних наук та інженерії Східноукраїнського національного університету ім. В. Даля, e-mail: maksym.nesterov@gmail.com

Бакитько Денис Едуардович – магістр кафедри комп'ютерних наук та інженерії Східноукраїнського національного університету ім. В. Даля, e-mail: bakitko_denis@ukr.net

Михайлова Аліса Олександрівна – магістр кафедри комп'ютерних наук та інженерії Східноукраїнського національного університету ім. В. Даля, e-mail: alicedemoran@gmail.com

Стаття подана 26.07.2019.