*Proceedings of the 10th International Conference on Applied Informatics*
*Eger, Hungary, January 30–February 1, 2017. pp. 31–39*
doi: 10.14794/ICAI.10.2017.31

# Efficiency test of Microsoft SQL Server 2016

**Tamás Balla, Tibor Radványi, Sándor Király, Roland Király**

Eszterházy Károly University
balla.tamas@uni-eszterhazy.com, radvanyi69@gmail.com
kiraly.sandor@uni-eszterhazy.hu, kiraly.roland@uni-eszterhazy.hu

**Abstract**

Databases play more and more important role in the information society.In every part of life, for example industry, government or education, it is extremely important that data can be stored as efficiently as possible, and queries run as fast as possible.

In this article, we summarize the results of the efficiency test performed on the Microsoft SQL Server 2016 database management system for the field of queries. We examine in what way and what large the efficiency methods increase the efficiency of queries. In this paper, we discuss how the various proposed techniques and methods influence the efficiencyin the terms of execution time, CPU and memory, and how the efficiency indicators change due to the increasing number of records.

We shed light on the efficiency issues of SQL queries: what large efficiency gains the variety of query optimization techniques cause depending on the stored data type and the number of records.

*Keywords:* database efficiency, SQL, query optimization techniques

*MSC:* 68P10, 68P20

## 1. Intorduction

The role of databases gradually gain importance in the information society since the data created swiftly and in big volumes are to be stored in such a way that can be later accessed. Nowadays, databases and client software based and run on them are applied in walks of life. One shall consider for instance the public administration, banking, healthcare, industry, and education. They share one thing for sure: they use databases and database systems with high volumes of records. The primary aim is that these databases shall be effective.

The effective database may have multiple meanings. It can be fast from the perspective how fast it executes the SQL commands; however, at the same time,

one shall remember that this is not the only point of view. Just consider storage spaces and processor time that can be the measuring tool for the efficacy of background processes. Regardless which database managing system is concerned, special attention shall be paid on it [1]. Kevin Kline et al. made the following assumption: *"According to our experience (and many professionals agree with us), the* 80% *of the performance increase regarding the SQL servers is due to the patches on SQL source codes and not to the smart change of settings or the adjustment of the operation system."*

Our aim is to investigate the efficacy of SQL commands prepared by applying different methods, techniques and approaches, and how they affect efficacy. It is not only measured from the perspective of quick response time but also the measuring on storage space and processor time will be attached.

## 2. Executing SQL commands

The SQL commands are translated by the query translator, hence the database engine may run them. Prior to translation, a syntactic analysis is required that checks whether the commands are elaborated in accordance with the rules of the language. If this analysis is over, the process of query translation is initiated.

The activity of query translator may be broken down into three major steps:

1. analysis,
2. rewriting of query,
3. elaboration of physical plan.

Query transcript and elaboration of physical plan are called query optimization. During the analysis, the query translator creates an analysis tree that is in accordance with the structure of the command. The transcript of query is carried out subsequent to successful analysis. In this step, the translator prepares an initial query plan from the analysis tree that is later transformed, if necessary, that the execution time is decreased and the results of the query remain the same. During the preparation of a physical plan, the logic query plan is converted to physical query plan by the translator [2, 3].

## 3. Sample database

To measure the efficacy of a SQL server, a database shall be established on which the query based on different techniques, approaches and methods may be measured. The sample database will be applied to prove or contradict in practice the theoretical efficacy boosting techniques based on the operation of the server.

Since we aim at measuring the efficacy of the database in multiple situations, we strived for

- establishing a database that is abundant in both data types and record types;
- that tables making up the database have one-more and more-more connections strings.

The elaborated database model includes the phone and internet records, clients, subscriptions, charges, data traffic and tariffs of a telecommunications company. The database model is displayed in Figure 1.

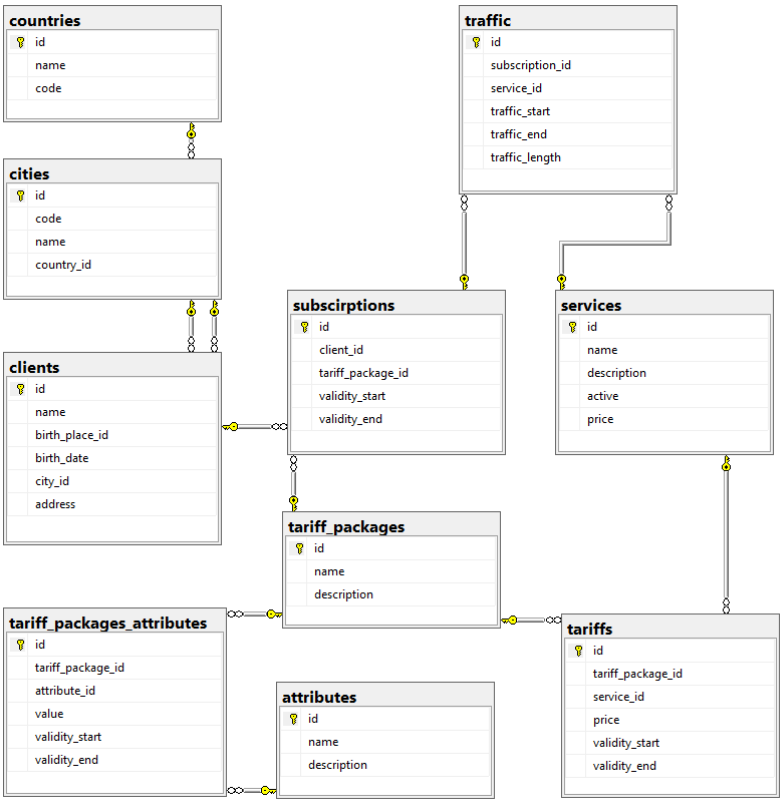


Figure 1: Sample database

# 4. Examinations

The main target of the examinations is to decide how the efficacy of queries may be increased by using the logical thinking during the elaboration of the queries and the techniques provided by the server. Our aim is not only setting up theories about the efficacy increase, but also testing it in practice.

As for the first step, we shall examine the queries during which we want the query the entire data content of certain tables. When querying all the fields of a table, one shall either list the fields of the table after SELECT or hit $*$ after the keyword. Let us examine if there is a difference between the methods regarding efficacy. Let us see the two queries to be tested.

```
SELECT traffic.*, services.*, clients.*
FROM traffic
        INNER JOIN services
                ON traffic.service_id = services.id
        INNER JOIN subscriptions
                ON subscriptions.id = traffic.subscription_id
        INNER JOIN clients
                ON clients.id = subscriptions.client_id
```

Query 1: SELECT *

```
SELECT traffic.id, traffic.subscription_id, traffic.service_id, traffic.
    traffic_start, traffic.traffic_end, traffic.traffic_length, services.
    id, services.name, services.description, services.active, services.
    price, clients.id, , clients.name, clients.birth_place_id, clients.
    birth_date, clients.city_id, clients.address
FROM traffic
        INNER JOIN services
                ON traffic.service_id = services.id
        INNER JOIN subscriptions
                ON subscriptions.id = traffic.subscription_id
        INNER JOIN clients
                ON clients.id = subscriptions.client_id
```

Query 2: SELECT field list

Subsequent to viewing the queries, let us examine that the series of queries on the sample database, namely, which method is more adequate for establishing an efficient system.

| **Records** | **CPU time (ms)** | | **Execution time (ms)** | |
|---|---|---|---|---|
| | **Query 1** | **Query 2** | **Query 1** | **Query 2** |
| **100 000** | 789.3 | 745.7 | 2 174.9 | 2 107.1 |
| **200 000** | 1 570.9 | 1 544.3 | 5 204.7 | 4 404.6 |
| **300 000** | 2 400.9 | 2 226.2 | 6 310.0 | 6 678.8 |
| **400 000** | 2 292.1 | 3 132.5 | 9 001.6 | 9 021.2 |
| **500 000** | 4 865.7 | 4 867.3 | 11 464.6 | 11 283.9 |

Table 1: CPU and exection time of SELECT * vs. SELECT field list (ms)

According to the table based on the examination, although there are certain minor efficacy differences between the two methods, no major dissimilarity is observed. This is rather strange as many works suggest refraining from applying the $*$ option as the using of keyword deteriorates efficacy. Based on the examination, it causes no problems in SQL Server 2016.

In the second examination, we investigate the effect of unnecessary use of DISTINCT keyword on efficacy. Only the individual hits are displayed among set of records at the output of the query. The decrease of efficacy is inevitable as the records shall be submitted to further examination. Let us prepare a query that queries all the traffic launched by the clients of the telecommunication company.

```
SELECT DISTINCT traffic.traffic_start, traffic.traffic_end, DATEDIFF(
    SECOND, traffic.traffic_start, traffic.traffic_end), services.name,
    services.price, clients.name, cities.name , clients_birth_date, lp.
    name, clients.address
FROM traffic
        INNER JOIN services ON traffic.service_id = services.id
        INNER JOIN subscriptions ON subscriptions .id = traffic.
            subscription_id
        INNER JOIN clients ON clients.id = subscriptions.client_id
        INNER JOIN cities lp ON lp.id = clients.city_id
        INNER JOIN cities ON cities.id = clients.birth_place_id
```

Query 3: SELECT with DISTINCT

```
SELECT DISTINCT traffic.traffic_start, traffic.traffic_end, DATEDIFF(
    SECOND, traffic.traffic_start, traffic.traffic_end), services.name,
    services.price, clients.name, cities.name , clients_birth_date, lp.
    name, clients.address
FROM traffic
        INNER JOIN services ON traffic.service_id = services.id
        INNER JOIN subscriptions ON subscriptions .id = traffic.
            subscription_id
        INNER JOIN clients ON clients.id = subscriptions.client_id
        INNER JOIN cities lp ON lp.id = clients.city_id
        INNER JOIN cities ON cities.id = clients.birth_place_id
```

Query 4: SELECT without DISTINCT

In such case, the data of the client, traffic and subscription are shown. It is trivial that none of the elements of the sets of result at the output of the query may identical, since only one traffic may be generated at one subscription at one time.

| **Records** | **CPU time (ms)** | | **Execution time (ms)** | |
|---|---|---|---|---|
| | **Query 3** | **Query 4** | **Query 3** | **Query 4** |
| **100 000** | 1 499.1 | 1 205.9 | 6 205.9 | 1 912.3 |
| **200 000** | 2 981.2 | 2 597.5 | 15 493.6 | 4 081.2 |
| **300 000** | 4 243.2 | 3 761.2 | 22 307.1 | 6 168.6 |
| **400 000** | 5 836.0 | 5 026.4 | 30 333.5 | 8 421.1 |
| **500 000** | 8 243.1 | 6 923.3 | 48 108.1 | 10 377.8 |

Table 2: CPU and exection time of SELECT DISTINCT vs. SELECT (ms)

In accordance with the experience gained (in Table 2.), it suggested to consider the use the DISTINCT keyword as the keyword significantly deteriorates efficacy. Initially, the running speed is double, but it triples in parallel with the increase of record numbers.

In the next examination, we observed the consequences of WHERE filter criteria. Upon the relative familiarity of the operation and values of records stored in particular tables, an SQL query may be altered, hence the number of records to be examined by the server is minimal. In such cases, the sequence of filter criteria is modified and the providing of trivial values is omitted. Let us observe the following query:

```
SELECT clients.name, tariff_packages.name, services.name, traffic.
    traffic_start, traffic.traffic_end, DATEDIFF(SECOND, traffic.
    traffic_start , traffic.traffic_end) as 'length'
FROM traffic
INNER JOIN subscriptions ON traffic.subscription_id = subscriptions.id
INNER JOIN tariff_packages ON tariff_packages.id = subscriptions.
    tariff_package_id
INNER JOIN services ON services.id = traffic.service_id
INNER JOIN clients ON clients.id = subscriptions.client_id
INNER JOIN cities ON cities.id = clients.city_id
WHERE '2004-01-01 00:00:00'<= traffic.traffic_start AND traffic.
    traffic_start <= '2004-12-31 23:59:59' AND '2004-01-01 00:00:00' <=
    traffic.traffic_end ND traffic.traffic_end <= '2004-12-31 23:59:59'
    AND traffic_length < 3600 AND services.name = 'call' AND cities.name =
     'Kismaros' AND clients.name = 'Kiss Kelemen'
```

Query 5: Not optimal query (AND operator)

Why the query is not efficient? Because not the most selective criterion is at first place of the WHERE criterion of the query, hence the evaluation of records is slower. Let us consider for a moment how many clients use a particular service at a telecommunication company annually and how many clients it has under the name of Kelemen Kiss who resides at Kismaros. An optimally designed query is as follows:

```
SELECT clients.name, tariff_packages.name, services.name, traffic.
    traffic_start, traffic.traffic_end, DATEDIFF(SECOND, traffic.
    traffic_start , traffic.traffic_end) as 'length'
FROM traffic
INNER JOIN subscriptions ON traffic.subscription_id = subscriptions.id
INNER JOIN tariff_packages ON tariff_packages.id = subscriptions.
    tariff_package_id
INNER JOIN services ON services.id = traffic.service_id
INNER JOIN clients ON clients.id = subscriptions.client_id
INNER JOIN cities ON cities.id = clients.city_id
WHERE
        clients.name = 'Kiss Kelemen' AND
        cities.name = 'Kismaros' AND
        services.name = 'call' AND
        '2004-01-01 00:00:00'<= traffic.traffic_start AND traffic.
            traffic_start <= '2004-12-31 23:59:59' AND '2004-01-01
            00:00:00' <= traffic.traffic_end ND traffic.traffic_end <= '
            2004-12-31 23:59:59' AND traffic_length < 3600
```

Query 6: Optimal query (AND operator)

Based on the described two queries, we shall take the results of the test measuring into consideration. The execution of queries is rather fast, however their execution time differs. The execution of the logically built queries is far faster. The particular criteria may be joined by AND operators in a criterion system. Moreover, the operation of the operator is known. That is to say, in case of phrases joined by AND operators, the most selective phrase shall be at the first place.

| Records | CPU time (ms) | | Execution time (ms) | |
|---|---|---|---|---|
| | Query 5 | Query 6 | Query 5 | Query 6 |
| **100 000** | 11.0 | 6.2 | 19.3 | 7.2 |
| **200 000** | 17.2 | 15.5 | 23.3 | 18.4 |
| **300 000** | 26.6 | 21.6 | 34.9 | 26.7 |
| **400 000** | 45.2 | 25.0 | 50.3 | 30.0 |
| **500 000** | 48.1 | 37.4 | 52.2 | 38.6 |

Table 3: CPU and exection time of not optimal and optimal queries (AND operator) (ms)

Now we shall consider the result of running the queries in Table 3. It is clear that significant increase in efficacy was gained by the optimization of the WHERE criterion. Moreover, we suppose that of the criterion is defined in accordance with the operation of the joining operators, the efficacy should be improved further.

Another operator for joining criteria is OR operator. In the case of this operator, the value of set of phrases joined by it is true if the value of any of the set of phrases is true. The building strategy is the opposite that of applied regarding AND operator: the phrase or criterion shall be at the first place that is met by the most record of the queried table. Upon creating a simple query, a criterion or a criterion system may be specified at WHERE clause that shall be met the elements of set of records created at the output of query, so they will be included in the set of records of the query. It is suggested to use as simple and as few criteria as possible, hence the query will be the most effective. Furthermore, criteria should be contracted in a way the criterion system remains logically equivalent. Moreover, trivial criterion shall be avoided as unnecessary evaluation deteriorates the efficacy. The unambiguous criteria shall be eliminated in the code of the query.Let us create a query that defines the traffic volume of 2010 of the telecommunications company. During the query, only traffic shorter than one hour shall be taken into consideration. But first let us take a query with a true output but it is not very effective as it comprises unnecessary criteria.

```
SSELECT clients.name, tariff_packages.name, services.name, traffic.
    traffic_start, traffic.traffic_end, DATEDIFF(SECOND, traffic.
    traffic_start , traffic.traffic_end) as 'length'
FROM traffic
INNER JOIN subscriptions ON traffic.subscription_id = subscriptions.id
INNER JOIN tariff_packages ON tariff_packages.id = subscriptions.
    tariff_package_id
INNER JOIN services ON services.id = traffic.service_id
INNER JOIN clients ON clients.id = subscriptions.client_id INNER JOIN
    cities ON cities.id = clients.city_id
WHERE 0 < traffic.length AND '2004-01-01 00:00:00' <= traffic.
    traffic_start AND traffic.traffic_start <= '2004-12-31 23:59:59' AND '
    2004-01-01 00:00:00' <= traffic.traffic_end ND traffic.traffic_end <=
    '2004-12-31 23:59:59' AND traffic_length < 3600
```

Query 7: Not optimal query (unnecessary conditions)

It is clear that set of records at the output of the query meet the above specified

conditions, however, it may be simplified, therefore made more effective. Let us take a simplified query in which the first three criteria are omitted, since the length of traffic will be more than 0. Moreover, it is unnecessary to restrict the end of the traffic from below and the beginning form above, since the other criteria guarantee that. Let us view the optimal query.

```
SSELECT clients.name, tariff_packages.name, services.name, traffic.
    traffic_start, traffic.traffic_end, DATEDIFF(SECOND, traffic.
    traffic_start , traffic.traffic_end) as 'length'
FROM traffic
INNER JOIN subscriptions ON traffic.subscription_id = subscriptions.id
INNER JOIN tariff_packages ON tariff_packages.id = subscriptions.
    tariff_package_id
INNER JOIN services ON services.id = traffic.service_id
INNER JOIN clients ON clients.id = subscriptions.client_id INNER JOIN
    cities ON cities.id = clients.city_id
WHERE '2004-01-01 00:00:00' <= traffic.traffic_start AND traffic.
    traffic_end <= '2004-12-31 23:59:59' AND traffic_length < 3600
```

Query 8: Optimal query (only necessary conditions)

After creating the two queries, We shall focus on their efficacy. It is expected that the efficacy of the query is will be improved. The efficacy examination is carried out in the ordinary way and the results are included in the following table and figure.

| **Records** | **CPU time (ms)** | | **Execution time (ms)** | |
|---|---|---|---|---|
| | **Query 7** | **Query 8** | **Query 7** | **Query 8** |
| **100 000** | 104.5 | 71.8 | 215.8 | 183.7 |
| **200 000** | 179.4 | 165.6 | 388.2 | 330.4 |
| **300 000** | 288.6 | 254.3 | 504.1 | 440.8 |
| **400 000** | 368.2 | 302.6 | 578.7 | 549.2 |
| **500 000** | 412.2 | 399.4 | 746.8 | 623.8 |

Table 4: CPU and exection time of not optimal and optimal queries (conditions) (ms)

It is clear that, in accordance with the expectations, the efficacy has been improved upon the simplification of the criterion system. The demonstrated case was an extreme example as the unnecessary criteria where located at the WHERE clause, hence they were evaluated in each case. It is not sure, however, the efficacy would have been improved if such criteria were at the end of the criteria sequence. It could have occurred that there is no need to evaluate the unnecessary criteria after the evaluation of criteria due to the evaluation of criteria at the beginning of the criterion system, hence the efficacy probably would not have been impacted. It is clear from the figure that the efficacy of optimized query is improved in line with the increasing record numbers

# 5. Conclusion

We have introduced how queries may be transformed and optimized to be the most effective that possible also the server during the execution of the command. The examinations have proved the effective management of the database requires not only the knowledge of SQL language, but also logic thinking and the consideration of statistical information. It is important think in advance how frequently certain attribute values appear in the given table. It is experienced that independent form auxiliary methods implemented on database servers, well structured and elaborated queries still play an important role.

# References

[1] Peter Gulutzan, Trudy Pelzer: SQL Performance Tuning (1st Edition), 2003., ISBN: 0201791692

[2] Jefrey D. Ullman, Jenifer Widom: A First Course in Database Systems (3rd Edition), 2009.

[3] Sándor Gajdos: Databases, Polytechnic University, 2006.