Data Administration in Information Systems

Lab 9: Index Tuning

In this lab class we will approach the following topics:

1. Index Tuning

In terms of tuning, the option that produces maximum gains with least impact on existing systems and processes is to examine your indexing strategy. However, the task of identifying the right indexes is not necessarily straightforward. It requires a sound knowledge of the sort of queries that will be run against the data, the distribution of that data, and the volume of data, as well as an understanding of what type of index will best suit your needs. Consider the following query:

```
SELECT A, COUNT(*)
FROM T
WHERE X < 10
GROUP BY A;
```

The following different physical design structures can reduce the execution cost of this query:
(i) A clustered index on X;
(ii) Table range partitioned on X;
(iii) A non-clustered index with key X and including the additional attribute A;
(iv) A materialized view that matches the query,
and so on.

These alternatives can have widely varying storage and update characteristics. Thus, in the presence of storage constraints, or for a workload containing updates, making a global choice for a workload is difficult. For example, a clustered index on a table and horizontal partitioning of a table are both non-redundant structures (i.e., they incur negligible additional storage overhead) whereas non-clustered indexes and materialized views can be potentially storage intensive and involve higher update costs. However, non-clustered indexes and materialized views can often be much more beneficial than a clustered index or a horizontally partitioned table. Clearly, a physical design tool that can give an integrated physical design recommendation can greatly reduce or even eliminate the need for a DBA to make ad-hoc decisions.

While understanding the basics is still essential, SQL Server does offer a helping hand in the form of some tools – in particular, the **Database Engine Tuning Advisor** – that can help to determine, tune and monitor your indexes. It can be used to get answers to the following questions:

- Which indexes are needed for specific queries?
- How to monitor index usage and its effectiveness?
- How to identify redundant indexes that could negatively impact performance?
- As the workload changes, how to identify missing indexes that could enhance performance for the new queries?

1.1. Using the Database Engine Tuning Advisor

Determining exactly the right indexes for your system can be quite a taxing process. For example, you have to consider:

- Which columns should be indexed, based on the knowledge on how the data is queried.
- Whether to choose a single-column index or a multiple column index.
- Whether to use a clustered index or a non-clustered index.
- Whether one could benefit from an index with included columns.
- How to utilize indexed (i.e. materialized) views.

Moreover, once you have determined the perfect set of indexes, your job is not finished. Your workload will change over time (i.e., new queries will be added, and older ones removed) and this might warrant revisiting existing indexes, analyzing their usage and making adjustments (i.e., modifying or dropping existing indexes and creating new ones). Maintenance of indexes is critical to ensure optimal performance in the long run.

The **Database Engine Tuning Advisor (DTA)** is a physical design tool providing an integrated console where DBAs can tune all **physical design features** supported by the server. The DTA takes into account all aspects of performance that the query optimizer can model, including the impact of multiple processors, amount of memory on the server, and so on. It is important to note, however, that query optimizers typically do not model all the aspects of query execution (e.g., impact of indexes on locking behavior, impact of data layout etc.). Thus, DTA's estimated improvement can be different from the actual improvement in execution time.

Taking as input a workload to fine-tune, i.e. a set of SQL statements that execute against the database server, the DTA produces a **set of physical design recommendations**, consisting of **indexes, materialized views, and strategies for horizontal range partitioning of tables, indexes and views**. The basis of DTA's recommendations is a *what-if* analysis provided by the SQL Server query optimizer, which allows the computation of an estimated cost as if a given configuration (e.g. the existence of some indexes) was materialized in the database. Similarly, to the actual evaluation of a given query plan, the query optimizer component can do an evaluation considering the *what-if* existence of a given physical design structure.

You can tune a single query or the entire workload to which your server is subjected to. A workload can be obtained, for instance, by using **SQL Server Profiler**, i.e., a tool for logging events (e.g. queries) that execute on a server. In this case, the workload would be given to the DTA in the form of a trace file, obtained with the SQL Server Profiler. The Profiler tool is just used to collect the workload, whereas the DTA performs the actual analysis and the tuning suggestions.

Alternatively, a workload can be specified as an SQL file containing an organization or industry benchmark. In this case, a text file with the SQL for each query in the workload would be given to the DTA.

The DTA can also take as input workloads referring to either a single or to a set of databases, as many applications use more than one database simultaneously.

Based on the options that you select, you can use the DTA to make recommendations for several Physical Design Structures (PDS), including:

- Clustered indexes
- Non-clustered indexes
- Indexes with included columns (to avoid bookmark lookups[1])
- Indexed views
- Partitions

The first step is to collect a workload for DTA to analyze. You can do this in one of two ways:

- **Using the Management Studio –** If you need to optimize the performance of a single query, you can use Management Studio to provide directly an input to DTA. Type the query in Management Studio, highlight it and then right click on it to choose Analyze in Database Engine Tuning Advisor.

- **Using the Profiler –** If you want to determine the optimum index set for the entire workload, corresponding to the actual queries that are being executed against an SQL Server instance, you should collect a profiler trace with the **TUNING template** (i.e. one of the possible options for the trace file that is generated by the SQL Server Profiler, and that contains all the information that is required by the Database Engine Tuning Advisor).

To fully exploit the effectiveness of DTA, you should always use a representative profiler trace. For instance, the indexes and partitioning considered by the DTA are limited only to interesting column groups (i.e., those columns that appear in a large fraction of the queries in the workload that have the highest cost) in order to improve scalability with little impact on quality. If the profiler trace is not representative of a true workload, important queries will likely be missing.

You should make sure that you subject your server to all the queries that will typically be run against the data, while you are collecting the trace. This could lead to a huge trace file, but that is

[1] A bookmark lookup is the retrieval of the actual data pages (the table data itself) once an entry has been found in a non-clustered index.

normal. If you simply collect a profiler trace over a 5-10 minute period, you can be pretty sure it will not be truly representative of all the queries executed against your database.

In SQL Profiler, the *TUNING* template captures only minimal events, so there should not be any significant performance impact on your server. A technique for workload compression is also employed, partitioning workloads with basis on a signature of each query (i.e., two queries have the same signature if they are identical in all aspects except for the constants referenced in the query).

2. Experiments and Exercises

For this lab, we are providing a workload (**lab9.sql**) against the **AdventureWorks** database. We recommend that you use this workload to get a hands-on perspective of the **Database Engine Tuning Advisor** (DTA). Alternatively, to obtain an SQL script with the workload, you can use the **SQL Profiler** to gather a system trace, and provide this trace as input to the DTA.

Assuming that the given workload is representative of the queries that would be executed against the database, you can use it as an input to the DTA, which will then generate recommendations. We will perform two types of analysis: (1) keeping the existing physical design structures, and (2) ignoring the existing physical design structures.

2.1. Keeping the Existing Physical Design Structures

This type of analysis is common and is useful if you have previously established the set of indexes that you deem to be most useful for your given workload, and are seeking further recommendations. To conduct this analysis:

a) Launch the **Database Engine Tuning Advisor** (DTA).

b) In the main window, under **Workload** select **File** and browse to the location of **lab9.sql**. This will be the input workload for the session.

c) In **Database for workload analysis**, select the AdventureWorks database.

d) In **Select databases and tables to tune**, check the AdventureWorks database in order to select all tables.

e) Change to the **Tuning Options** tab.

f) Uncheck the option **Limit tuning time**.

g) In **Physical Design Structures (PDS) to use in database**, select **Indexes and indexed views**.

h) In **Physical Design Structures (PDS) to keep in database**, select **Keep all existing PDS**.

i) In the toolbar, hit the **Start Analysis** button.

Once DTA finishes consuming the workload, it will list its recommendations under the **Recommendations** tab.

j) Hover the mouse over the **Target of Recommendation** column in order to inspect some of the recommendations in more detail.

2.2. Ignoring the Existing Physical Design Structures

In the previous scenario, DTA makes recommendations for any missing indexes. However, this does not necessarily mean your existing indexes are optimal for the query optimizer. You may also consider conducting an analysis whereby DTA ignores all existing physical design structures and recommends what it deems the best possible set for the given workload. This way, you can validate your assumptions about what indexes are required.

a) In the toolbar, click the **Start New Session** button.

b) Under **Workload**, browse to the file **lab9.sql**.

c) In **Database for workload analysis**, select the AdventureWorks database.

d) In **Select databases and tables to tune**, check the AdventureWorks database.

e) Change to the **Tuning Options** tab.

f) Uncheck the option **Limit tuning time**.

g) In **Physical Design Structures (PDS) to use in database**, select **Indexes and indexed views**.

h) In **Physical Design Structures (PDS) to keep in database**, select **Do not keep any existing PDS**.

Contrary to how this might sound, the DTA will not actually drop or delete any existing physical design structures. This is the biggest advantage of using DTA, as it means you can use the tool to perform *what-if* analysis without actually introducing any changes to the underlying schema.

i) In the toolbar, hit the **Start Analysis** button.

After consuming the workload, DTA presents a set of tuning recommendations. A good idea is to focus on the following columns:

- **Recommendation –** this is the action that you need to take. Possible values include **create** or **drop**.

- **Target of Recommendation –** this is the proposed name of the physical design structure to be created. The naming convention is typical of DTA and generally starts with _dta*. However, it is recommended that you change this name based on the naming convention in your database.
- **Definition –** this is the list of columns that this new physical design structure will include. If you click on the hyperlink, it will open up a new window with the T-SQL script to implement this recommendation.
- **Estimated Improvements –** shown at the top of the **Recommendations** tab, this is the estimated percentage improvement that you can expect in your workload performance, if you implement all the recommendations made by DTA.
- **Space used by recommendation (MB) –** shown in the **Reports** tab, this is the extra space in MB that you would need, if you decide to implement these recommendations.

The **Reports** tab features several different reports. Here we highlight just three of these reports:

- **Statement cost report -** This report lists individual statements in your workload and the estimated performance improvement for each one. Using this report, you can identify your poorly performing queries and see the sort of improvement you can expect if you implement the recommendations made by DTA. You will find that some statements do not have any improvements (**Percent Improvement** = 0.00). This is because either the statement was not tuned for some reason or it already has all the indexes that it needs to perform optimally.
- **Index usage report (current) -** This report shows how the existing indexes are being used. Each index that has been used by a query is listed here. Each index has a **Percent Usage** value which indicates the percentage of statements in your workload that referenced this index. If an index is not listed here, it means that it has not been used by any query in your workload. If you are certain that all the queries that run against your server have been captured by your profiler trace, then you can use this report to identify indexes that are not required and possibly delete them.
- **Index usage report (recommended) –** This report shows how index usage will change if the recommended indexes are implemented. If you compare these two reports, you will see that the index usage of some of the current indexes has fallen while some new indexes have been included with a higher usage percentage, indicating a different execution plan for your workload and improved performance.

2.3. Implementing the Recommendations

By now, we got a set of recommendations to improve performance. You then have the choice to either:

- Save the recommendations – you can save the recommendations in an SQL script by using the menu option **Actions > Save Recommendations**. You can then manually run the script in

Management Studio to create all the recommended physical design structures.

- Apply the recommendations using the DTA – if you are happy with the set of recommendations, then select the menu option **Actions > Apply Recommendations**. You can also schedule a later time to apply these recommendations, for instance during off-peak hours so that interference with other operations is minimal.

Now, suppose you do not want to apply all the recommendations that the DTA provided. Since the **Estimated improvement** value can only be achieved if all the recommendations are applied, you are not sure what kind of impact there will be if you choose to apply only a subset of those recommendations. This is where performing a *what-if* analysis becomes very useful.

To do a *what-if* analysis, deselect the recommendations that you do not want to apply. Then select the menu option **Actions > Evaluate Recommendations**. This will launch another session with the same options as the earlier one. When you click on **Start Analysis**, DTA will provide the estimated performance improvements, based on just the selected subset of recommendations. Again, the important thing to remember is that DTA performs this *what-if* analysis without actually implementing anything in the database.

2.4. Exercise

Consider the following table,

employees(id, name, salary, department, contract_year)

which is subject to the following workload:

a) How many employees were hired in the current year?
b) Who are the employees (ids) with the highest salaries?
c) What is the total amount of salaries paid by each department?
d) What is the average number of employees per department?

For each query individually, which indices would you create over the table? Indicate the index column(s), whether it is clustered or non-clustered, and its type (B+tree or hash). Justify.