Database Tuning – Assignment 1

Uploading Data to the Database

A2

Baumgartner Dominik, 0920177

Dafir Thomas Samy, 1331483

Schörgnhofer Kevin, 1421082

October 18, 2016

Straightforward Implementation

Implementation For the Straightforward Implementation we create a single SQL-INSERT command for every line of the given input file. We send the current command to the database server, before we start creating the SQL-INSERT command for the next line of the input file.

```
INSERT INTO auth VALUES ('Jurgen Annevelink','books/acm/kim95/AnnevelinkACFHK95');
INSERT INTO auth VALUES ('Rafiul Ahad','books/acm/kim95/AnnevelinkACFHK95');
INSERT INTO auth VALUES ('Amelia Carlson','books/acm/kim95/AnnevelinkACFHK95');
etc.
```

Efficient Approach 1: Single Query

Implementation For this approach we build one huge SQL-INSERT command which contains all the values of the input file.

```
INSERT  INTO auth VALUES
('Jurgen Annevelink','books/acm/kim95/AnnevelinkACFHK95'),
('Rafiul Ahad','books/acm/kim95/AnnevelinkACFHK95'),
('Amelia Carlson','books/acm/kim95/AnnevelinkACFHK95'),
...
('Klaus Richter','series/vdi/SchenkR07');
```

Why is this approach efficient? There are two main reasons, why this approach is faster than the straightforward implementation:

1. Compared to the straightforward approach we have only about 60% of the amount of data which has to be sent via the network to the database server. In the

straightforward approach we have a separate "INSERT INTO auth VALUES"-statement for every single input line, which is about 40% of the whole INSERT-command (INSERT-command = INSERT-statement + values). In this approach we need the INSERT-statement only once, causing the size of the transferred data to shrink to about 60%. Less data needs less time to be sent.

2. For the straightforward approach we send about 3 million sql commands via the network to the database server, whereas for this approach we need only one. This cuts down the overhead for sending the sql command(s) vastly. Arrived at the database server, this time the database has to parse, optimize and create/access paths to the data only once (and not 3 million times), which again saves a lot of time.

This approach was already known to us and the reasons why this approach is faster are pretty obvious (which is the reason why there are no references for this approach).

Tuning principle "Start-up Costs Are High; Running Costs Are Low":
For this approach we tried to care for the **network latency** and cut down the **query overhead** with using only one single query.

Efficient Approach 2: COPY FROM statement

Implementation For the second improved approach we wanted to use the "COPY FROM"-statement to import the data directly from a file into the database. Unfortunately this statement requires superuser rights which we do not have. In fact we were not able to use this approach, which is the reason why we searched for a third approach.

```
COPY auth FROM '/home/stud3/%user%/Documents/dbtuning/blatt01/auth.tsv' DELIMITER '\tab';
```

Why is this approach efficient? As already mentioned we were not able to actually test this approach. But the following arguments lead to the assumption that it should be faster than the straightforward approach:

- The size of the data which has to be transferred via the network is pretty small. You have only the COPY-statement, the path to the document and the option "DELIMITER". Less data needs less time to be sent.
- There is only one sql command sent, and not about 3 million, which causes much less overhead and network traffic. Additionally the database has to parse, optimize and create/access paths to the data only once.
- In the straightforward approach the computer which executes the import program has to calculate all 3 million SQL-commands. In this approach the executing computer has to calculate nearly nothing. All the work is done by the database server, which probably has more computing resources.

- The COPY-statement is a statement which was made for fast imports (and exports) from files (or to files), so it is probably faster than a separate program which first reads data from a file and then sends it to the database as an INSERT-statement.[1]

Tuning principle "Start-up Costs Are High; Running Costs Are Low":
Like in our first improved approach there is **less network traffic** and a small **query overhead**.
"Render on the Server What Is Due on the Server": The combination of the import instruction and the values is not created on client side, but on server side.

Efficient Approach 3: (Prepared Statement)

Implementation JDBC and the *PreparedStatement* class included in Java were used in this approach. The connection is established through JDBC. This results in a connection object which can then be used to create a prepared statement:

```
String query = "INSERT INTO auth values(?, ?);";
PreparedStatement ps = connection.prepareStatement(query);
```

This statement is then sent directly to the database which compiles it. We then use the batch-insert functionality a *PreparedStatement* - object offers. All data rows are added to the batch and finally the query is executed through executing the prepared statement with the collected data.

Why is this approach efficient? It is efficient in two ways:

- It reduces the amount of data that needs to be sent to the database.
- It reduces the amount of query-statements which have to be compiled and optimized by the database.

This approach is made efficient using two tuning principles:

- Using precompiled statements: The query we want to execute is sent to the database, compiled and optimized right away. Once we collect all our data rows and send them to the database the precompiled statement is used directly without the need to go through the compilation and optimization process for every single row (in contrast to the straightforward approach). This saves huge amounts of time as can be seen in the experiments [2].
- Reducing network latency by reducing the amount of data sent to the database. The query is only sent once, after that only data rows are transmitted. Compared to the straight forward approach this saves a lot of data, reducing the total amount of data to send and thus increasing the total speed of the transmission [3].

[1] PostgreSQL 9.2 Documentation, Chapter VI.I SQL Commands, `https://www.postgresql.org/docs/9.2/static/sql-copy.html`
[2] JDBC Documentation,`https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html`
[3] Lecture Slides by Prof. Augsten

Tuning principles "Use precompiled statements" (here: prepared statements). "Reduce network latency" (through reducing amount of data to be transfered).

Runtime Experiment

Approach	Runtime [sec]
Straightforward	9999
Approach 1 (single query)	54
Approach 3 (prepared statement)	128

Setting

All experiments were executed in the computer room (on campus). A direct connection to the database server (biber) was established from there using Java 8 and JDBC.

Time Spent on this Assignment

Time in hours per person: **X**XX