Database Tuning – Assignment 1

Uploading Data to the Database

A3

Platzer Hugo, 1421579

Strohmeier Mario, 1422959

October 13, 2016

Straightforward Implementation

Implementation Every tuple gets inserted individually, by iterating over each line from the tsv file and executing the insert.

```
"INSERT INTO auth (name, pubid) values (?, ?)"
```

Efficient Approach 1: (Batch/Bulk Insert)

Implementation With this method, we don't execute the insert for each tuple seperatly. We keep adding tuples to a batch, until it is big enough or there are no tuples left. After the batch is big enough, we execute the insert for our whole batch.

```
"INSERT INTO auth (name, pubid) values (?, ?)"
```

Why is this approach efficient? This approach is more efficent, since we don't need to write a log for every single insert. This frees a lot of disk access time. Furthermore there is a lot less connection overhead, because the packets we send are bigger.[1]

Tuning principle start-up costs are high; running costs are low

Efficient Approach 2: (SQL COPY statement)

Implementation Our data is in the form of a TSV file meaning each line contains a (auth, pubid) tuple, the two values separated by a TAB character. Data like this can be directly loaded into the Postgres database by using the COPY command. This can be done using the psql prompt:

[1] `https://dbresearch.uni-salzburg.at/teaching/2016ws/dbt/dbt_01-handout-1x1.pdf`, Page 23-27

```
\copy auth from ~/dbtuning/data/auth.tsv DELIMITER e'\t'
```

It can also be done with JDBC using the CopyManager class and a FileReader for the input:

```
public static void insertCopyFrom(Connection con, String filename, int n)
        throws SQLException, IOException {
    CopyManager copyManager = new CopyManager((BaseConnection) con);
    FileReader fileReader = new FileReader(filename);

    copyManager.copyIn("COPY auth from STDIN WITH DELIMITER '\t'", fileReader);
}
```

Why is this approach efficient? Compared to the straightforward approach, this creates one "query" for the whole file, avoiding excessive network roundtrips and minimizing data overhead (query parameters). The database systems' insert code is only called once. Also it prevents lots of unneccessary logfile access. Most of the file/data processing is done on the server, this might also be more efficient than processing the file manually line per line in Java.

References:

- Postgres Documentation, COPY statement

  `https://www.postgresql.org/docs/9.2/static/sql-copy.html`

- Stackoverflow, 'How to copy data from file to Postgres using JDBC'

  `https://stackoverflow.com/questions/6958965/how-to-copy-a-data-from-file-to-postgresql-using-jdbc`

Tuning principle "Start-Up Costs Are High; Running Costs Are Low"

Runtime Experiment

Approach	Runtime [sec]
Straightforward	11400
Batch / Bulk Insert	120
SQL COPY statement	8

XXX

Notes:

- The provided "biber" database server was used as target for uploading the data, tsv files were stored on the cosy user account home folder, the program was run from a machine at the computer room of the facility.

- For the straightforward approach, only 10000 records were uploaded, we scaled the 38 sec. taken linearly to approximate the time taken for all ≈ 3 mio records.
- For the batch insert, we chose a batch size of 1000 since experiments showed larger batch sizes do not give further speedup.

Time Spent on this Assignment

Time in hours per person: **3.14159**