Assignment 1

Uploading Data to the Database

Database Tuning

Group Name 6
Pape David, 01634454
Vecek Filip, 11700962
Paulitsch Matthias, 01652394

20. März 2019

Experimental Setup

Bei unserer Ausarbeitung verwenden wir einen lokalen Server mit der Postgres Version 11.2. Die Datenbank hat zwei zu den gebenen Daten korrespondierene Relationen `Auth` und `Publ`, deren Attribute wie in der Angabe definiert sind. Wir haben uns auf Basis der von Ihnen bereitgestellten Java-Vorlage mit dem Server verbunden und in dieser Klasse die verschiedenen Ansätze implementiert. Vor jedem Durchlauf werden die von möglichen vorheringen Durchläufen eingefügten Daten mit

```
DROP TABLE IF EXISTS Auth;
DROP TABLE IF EXISTS Publ;
```

gelöscht.

Straightforward Implementation

Implementation Für den naiven Ansatz verwenden wir einen `BufferedReader`, mit dem wir die Dateien zeilenweise in einen String einlesen. Der Sring wird dann mit `split("\t")` in die Werte für die Attribute aufgeteilt, welche mit dem SQL-Befehl

```
INSERT INTO Auth (name, pubId) VALUES ('%s', '%s')
```

einzeln in die `Auth`-Relation (und analog in `Publ`) eingefügt werden.

Efficient Approaches

Efficient Approach 1: No Autocommit

Implementation Für den ersten effizienten Ansatz haben wir Autocommit ausgeschaltet, während die Einträge in die Datenbank eingefügt wurden. Wenn eine Änderung commited wird, ist garantiert, dass sie für andere Nutzer sichtbar ist und im Falle eines Crashes bestehen bleibt (die Änderung wird garantiert geschrieben).

Wir haben Autocommit ausgeschaltet, indem wir den naiven Ansatz in zwei SQL-Statements "eingewickelt" haben:

```
BEGIN;
-- execute naive approach...
COMMIT;
```

Why is this approach efficient? Wenn autocommit eingeschaltet ist, wird nach jedem insert ein commit durchgeführt. Jeder dieser commits benötigt eine eigene Synchronisation, aller ausstehenden Schreibvorgänge, bevor die Anfrage erfolgreich abgeschlossen werden kann. Dies hindert das Betriebssystem daran, die Schreibvorgänge mehrerer statements, intelligenter abarbeiten zu können. Weiters kann dadurch die I/O Effizienz vom Betriebssystem nicht verbessert werden. Zusätzlich werden vor jedem Statement Pausen eingefügt, bevor diese etwas returnen können.

Durch das Ausschalten von autocommit, fallen alle diese Performance Nachteile weg. Jedoch hat auch diese Optimierungstechnik einen Nachteil, nämlich dass die Daten nicht mehr sofort gesichert werden. [1]

Tuning principle Dieser Ansatz folgt dem "Start-Up Costs Are High; Running Costs Are Low"-Prinzip. Mit Autocommit wird nach jedem Insert-Statement auf die Platte geschrieben, während die Schreibvorgänge ohne Autocommit gepuffert werden können. Die "Start-Up Costs" liegen in der Vielzahl an einzelnen Schreibzugriffen, die durch Autocommit generiert werden.

In unserem Experiment legt die Datenbank ihre Daten auf einer SSD ab, weswegen der Unterschied zwischen Autocommit und No Autocommit wahrscheinlich kleiner ist, als er mit einer Festplatte wäre, da diese für einzelne Schreibvorgaenge mehr Zeit brauchen.

Das Prinzip "Be Prepared For Trade-Offs" tritt hier auch auf, da man, wie bereits erwähnt, die Sicherheit verliert, dass die Updates persistent gesichert sind.

Efficient Approach 2: Copy-Statement

Implementation In unserem zweiten effizienten Ansatz verwenden wir das `Copy`-Statement von SQL. Dazu haben wir nur das folgende Kommando verwendet:

```
COPY Auth (name, pubId) FROM 'PATH_TO_AUTH';
```

Why is this approach efficient? Im Vergleich zu den beiden vorherigen Ansätzen werden viele Kosten eingespart, die bei der Ausführung vieler Anfragen entstehen, darunter:

- Latenzen, die bei der Übertragung vieler Anfragen über ein Netzwerk entstehen
- Kosten, die durch das Parsen und Planen von SQL-Statements entstehen
- Commit-Kosten (wie beim ersten effizienteren Ansatz) [2]

Tuning principle Hier tritt auch das "Start-Up Costs Are High; Running Costs Are Low"-Prinzip auf. Die eingesparten "Start-Up Costs" liegen in den oben genannten

[1] https://www.postgresql.org/message-id/4D9E6D59.708@postnewspapers.com.au
[2] https://stackoverflow.com/questions/46715354/how-does-copy-work-and-why-is-it-so-much-faster-than-insert

Punkten. Hinzu kommen einige Optimierungen, die das `Copy`-Statement automatisch durchführt.

Runtime Experiment

Notes Wie bereits erwähnt läuft die Datenbank lokal. Die Daten sind auf einer SSD abgelegt und die Anfragen werden von einem Java-Client aus gemacht, der auf derselben Maschine läuft.

Approach	Runtime [sec]
Straightforward	1427,723
No Autocommit	606,845
Copy-Statement	30,259

Time Spent on this Assignment

Time in hours per person: **4**