

**Datenbanken und Informationssysteme**

**Dokumentation und Ausarbeitung der Messungen Transaktionsdurchsatz**

****

**Praktikumsaufgabe 9**

**Gruppe 2.6**

Bearbeiter : Jannik Ewers und Benedikt Oppenberg

Eingereicht am 18. Dezember 2013

Inhalt

[Aufgabe 1](#_Toc375169808)

[Dokumentation und Bewertung der Messergebnisse für 1 remote Client 2](#_Toc375169809)

[Dokumentation und Bewertung der Messergebnisse für 5 remote Clients 2](#_Toc375169810)

[Dokumentation und Bewertung der Messergebnisse für 5+5 remote Clients 3](#_Toc375169811)

[Optimierung im Programm 3](#_Toc375169812)

[Optimierung im DBMS 3](#_Toc375169813)

[Fazit zum DBMS 3](#_Toc375169814)

[Code 4](#_Toc375169815)

## Aufgabe

Die Praktikumsaufgabe war es, ein Java-Programm zu entwickeln, welches drei verschiedene Transaktionen mit der 50tps-Benchmark Datenbank durchführt.

(TX 1) Analyse-Transaktion:

Gib die Anzahl der Einträge in der Tabelle HISTORY zurück, die den gleichen Wert für DELTA besitzen.

(TX 2) Einzahlungs-Transaktion:

In der Relation BRANCHES soll die zu BRANCHID gehörige Bilanzsumme

BALANCE aktualisiert werden.

In der Relation TELLERS soll die zu TELLERID gehörige Bilanzsumme BALANCE

aktualisiert werden.

In der Relation ACCOUNTS soll der zu ACCID gehörige Kontostand BALANCE

aktualisiert und zurückgegeben werden

In der Relation HISTORY soll die Einzahlung protokolliert werden.

(TX 3) Kontostands-Transaktion:

Gib den Kontostand eines Kontos zurück

Diese drei Transaktionen sollen in dem Programm zufällig aufgerufen werden, mit einer Gewichtung von (TX1:35 zu TX2:50 zu TX3:15). Die Funktionen sollen zehn Minuten lang hintereinander aufgerufen werden, mit einer Nachdenkzeit von 50ms dazwischen.

Gemessen werden soll fünf Minuten lang, beginnend nach vier Minuten Einschwingphase.

In dieser Zeit wird die Anzahl der ausgeführten Transaktionen gezählt.

Es soll in drei Phasen gemessen werden.

In der ersten Phase, soll von einem Client aus gemessen werden.

Phase zwei, besteht darin, auf einem Rechner fünf Clients zu starten.

In der letzten Phase sollen auf zwei Rechnern von je fünf Clients aus gemessen werden.

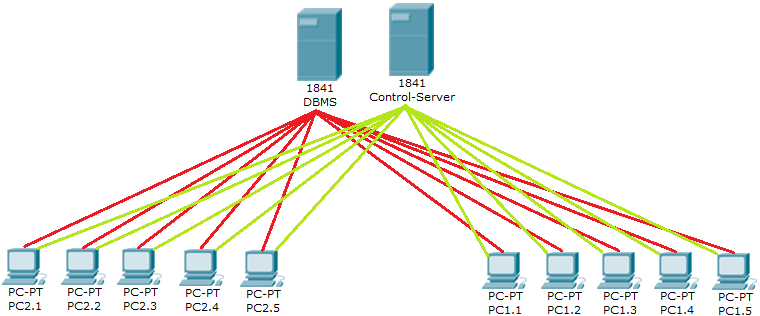
Als Grundlage für dieses Projekt wurde uns das Benchmark-Framework angeboten.

Wir entschieden uns aber dafür, ein eigenes Framework zu benutzen, welches wir auf der Grundlage des in Praktikumsaufgabe 7 verwendet haben aufzubauen.

Ziel sollte es sein, dass es einen Controller oder Server gibt, der die Tests bei den Clients startet und die Messergebnisse am Ende sammelt und Auswertet.

Alle Clients haben sowohl eine eigene Verbindung zu Datenbank, als auch zum Benchmark-Controller.

So ergibt sich folgendes Verbindungsmodell:



Damit wir nicht mehrere Programme entwickeln müssen haben wir uns dafür entschieden, den Server und den Client in einem zu entwickeln.

Dies erreichen wir, indem das Programm selbst zum Server wird, sobald unter der angegeben Adresse kein Server erreicht werden kann.

## Dokumentation und Bewertung der Messergebnisse für 1 remote Client

Bei der ersten Messung mit einem Remote Client erzielten wir einen Wert von 15 tps

Nach einem Optimierungsschritt erzielten wir 15,59 tps

Nach weiteren Optimierungsversuchen erzielten wir dann einen Messwert von 15,92 tps

## Dokumentation und Bewertung der Messergebnisse für 5 remote Clients

Bei der ersten Messung mit einem Remote Client erhielten wir einen Wert von 76 tps

Nach einem Optimierungsschritt erzielten wir 77,88 tps

Nach weiteren Optimierungsversuchen erzielten wir dann einen Messwert von 79 tps

Bei der Betrachtung der Messergebnisse fällt auf, dass die Werte um den Faktor 5 gestiegen sind. Also hat jede Datenbankverbindung immer ~ 15 tps.

## Dokumentation und Bewertung der Messergebnisse für 5+5 remote Clients

Bei der ersten Messung mit einem Remote Client erzielten wir einen Wert von 159.09 tps

Nach einem Optimierungsschritt erzielten wir 159 tps

Nach weiteren Optimierungsversuchen erzielten wir dann einen Messwert von 158 tps

Auch hier fällt sofort auf, dass die Werte, wie erwartet auf das zweifach steigen, da doppelt so viele Clients ausgeführt werden.

## Optimierung im Programm

Im Load-Driver Programm wird bei der Einzahlungs-Transaktion, statt drei Änderungsabfragen , eine Einfügeabfrage und einer Abfrage, wird eine „stored procedure“ aufgerufen.

Diese führt festgelegten Code in der Datenbank aus und gibt mir den benötigten Wert zurück.

## Optimierung im DBMS

In der Datenbank wurde eine „stored procedure“ namens Einzahlungs\_TX() erstellt.

Die Prozedur ändert die BALANCE in BRANCHES, TELLERS, ACCOUNTS und erstellt ein HISTORY-Eintrag.

Danach gibt die Datenbank den neuen Wert in ACCOUNTS.BALANCE zurück.

Diese Änderung brachte ein bis zwei Transaktionen/s mehr.

Außerdem wurden in dem Datenbankmanagementsystem die Variable SQL\_LOG\_BIN auf null gesetzt.

Die Systemvariable SQL\_LOG\_BIN gibt an, ob jedes Event in der Datenbank, welches Daten verändert oder erstellt protokolliert wird oder nicht.

Diese Änderung brachte 0,5 bis eine Transaktion/s mehr.

## Fazit zum DBMS

Das Ziel der Praktikumsaufgabe ist es ein Load-Driver-Programm so zu schreiben, und das DBMS so zu konfigurieren, dass möglichst viele Transaktionen pro Sekunde durchgeführt werden können.

Unser bester Wert ist: 15.92 tps für einen Client. Wir sind mit diesen Wert zufrieden, denken

jedoch, dass wir diesen Wert noch steigern können, durch Beispielsweise Veränderungen der

MySQL Puffer Größen. Viele Optimierungsschritte, die wir durchgeführt haben waren

Gewinnbringend und andere hatten Augenscheinlich gar keine Veränderung bewirkt.

Wir hatten das Datenbankmanagementsystem MySQL, und waren damit auch sehr zufrieden.

Durch die große Nutzer- und Entwicklergemeinde hatten wir viele Ressourcen, auf die wir

zurückgreifen konnten. Auch hatten wir den Eindruck, dass MySQL nicht langsamer als vergleichbare DBMS oder instabil läuft.

Ein weiterer Vorteil von MySQL in Kombination mit der MySQL Workbench ist die leichte

Bedienung. Da MySQL Workbench hier eine ansprechende Grafische Oberfläche bietet, mit der die

meisten Operationen ohne Probleme durchgeführt werden können.

## 

## Code

**import** java.io.BufferedReader;

**import** java.io.DataInputStream;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.PrintStream;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Random;

**import** java.util.Scanner;

/\*\*

\* Dies ist der Load-Driver zur DBI-Praktikumsaufgbe 9

\* Er enhält den Client und Server

\*

\* **@author** Benedikt Oppenberg, Jannik Ewers

\* **@version** 1.0

\*/

**public** **class** Connect **implements** Runnable

{

//Global

**private** Boolean bRun;

**private** Thread runner;

**private** ServerSocket ss;

**private** List<Socket> Clients;

**private** List<InputStreamReader> Inputs;

**private** List<PrintStream> Outputs;

/\*\*

\* Konstruktor

\*/

**public** Connect() **throws** SQLException, ClassNotFoundException, IOException

{

//Init

bRun = **true**;

runner = **new** Thread(**this**);

Clients = **new** ArrayList<Socket>();

Inputs = **new** ArrayList<InputStreamReader>();

Outputs = **new** ArrayList<PrintStream>();

doSome();

}

/\*\*

\* Steuerung des Load-Drivers

\* Erzueugung des Clienten und/oder Servers

\*

\* throws SQLException Fehler beim Verarbeiten

\* throws IOException Socket IOException

\* throws ClassNotFoundException

\*/

**public** **void** doSome() **throws** SQLException, IOException, ClassNotFoundException

{

//Init local vars

**int** iTxCount = 0;

**int** iPercent = 0;

**long** tnow, tend, tbeg;

Boolean bIsServer = **false**;

java.sql.Statement stmt = **null**;

Socket Client = **null**;

Scanner scr = **new** Scanner(System.*in*);

//Init final vars

**final** **int** iGesamtZeit = 600000;

**final** **int** iEinschwingZeit = 240000;

**final** **int** iAusschwingZeit = 540000;

String user = "root";

String pass = "janbe2013";

String DB = "benchmark";

System.*out*.print("DB-Server : ");

String ConnectionName = "jdbc:mysql://"+scr.nextLine()+"/" + DB;

System.*out*.print("Data-Server: ");

**try**

{

//Versucht eine Verbindung zu dem Messdaten-Server aufzubauen

Client = **new** Socket(scr.nextLine(), 1235);

System.*out*.println("Connect as client");

}

**catch**(Exception e)

{

//Falls keine Verbindung hergestellt werden kann wird ein Server erstellt

System.*out*.println("Connect as server");

bIsServer = **true**;

ss = **new** ServerSocket(1235);

//Starte Thread

runner.start();

}

**if**(bIsServer)

{

//Warten bis alle Clients verbunden sind

System.*out*.println("Press enter to start...");

System.*in*.read();

//Thread schließen

bRun = **false**;

**new** Socket(ss.getInetAddress(), ss.getLocalPort()).close();

//Allen Clienten das Startsignal senden

**for**(PrintStream ps : Outputs)

{

ps.print("Go");

}

}

**else**

{

//Auf das Startsignal warten

DataInputStream ois = **new**

DataInputStream(Client.getInputStream());

ois.read();

System.*out*.println("\n");

}

System.*out*.println("Server sended go");

//Verbindung zum DB-Server herstellen

Connection con = DriverManager.*getConnection*(ConnectionName, user,

pass);

stmt = con.createStatement();

System.*out*.print("|1%\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_50%\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_100%|\n|");

//Tabelle history leeren

*trunc\_table*(stmt);

Random r = **new** Random();

tend = System.*currentTimeMillis*() + iGesamtZeit;

tnow = System.*currentTimeMillis*();

tbeg = tnow;

**while**(tnow < tend )

{

tnow = System.*currentTimeMillis*();

**switch**(*rand*())

{

**case** 1:

*Kontostand\_TX*(stmt, r.nextInt(10000)+1);

**break**;

**case** 2:

*Einzahlungs\_TX*(stmt, r.nextInt(10000)+1, r.nextInt(10000)+1, r.nextInt(10000)+1, r.nextInt(10000)+1);

**break**;

**case** 3:

*Analyse\_TX*(stmt, r.nextInt(10000)+1);

**break**;

}

**if**((tnow - tbeg) > iEinschwingZeit && (tnow - tbeg) < iAusschwingZeit)

iTxCount++;

**int** progress = (iGesamtZeit - (**int**)(tend-tnow))\*100/iGesamtZeit;

**if**(progress > iPercent)

{

**if**(progress%2 == 0)

System.*out*.print("=");

iPercent = progress;

}

//Nachdenkzeit

**try** {Thread.*sleep*(50);}**catch** (InterruptedException e) {e.printStackTrace();}

}

System.*out*.println("|\n\n");

**if**(bIsServer)

{

//Auf die Clienten Warten

**try** {Thread.*sleep*(5000);}**catch** (InterruptedException e){e.printStackTrace();}

**int** txCount = 0;

//Daten von den Clienten holen

**for**(InputStreamReader dis : Inputs)

{

BufferedReader bufferedReader = **new** BufferedReader(dis);

**char**[] buffer = **new** **char**[20];

**int** anzahlZeichen = bufferedReader.read(buffer, 0, 20);

String nachricht = **new** String(buffer, 0, anzahlZeichen);

txCount += Integer.*parseInt*(nachricht);

}

System.*out*.println("Cleints: " + (Clients.size()+1));

System.*out*.println("Tx : " + (iTxCount + txCount));

System.*out*.println("TxPS : "+ ((iTxCount + txCount) / ((iAusschwingZeit-iEinschwingZeit)/1000)) );

}

**else**

{

//Daten an den Server senden

PrintStream dot = **new** PrintStream(Client.getOutputStream());

dot.print(iTxCount);

System.*out*.println("Ended");

scr.next();

}

scr.close();

stmt.close();

con.close();

}

/\*\*

\* Wartet auf neue Clienten.

\* Wenn eine Verbindung zu einem Clienten hergestellt wurde,

\* wird er und seine Streams in separate Listen eingereiht

\*/

**public** **void** run()

{

**while**(bRun)

{

**try**

{

Clients.add(Clients.size(), ss.accept());

**if**(bRun)

{

Inputs.add(Inputs.size(), **new** InputStreamReader( Clients.get(Clients.size()-1).getInputStream()));

Outputs.add(Outputs.size(), **new** PrintStream( Clients.get(Clients.size()-1).getOutputStream()));

System.*out*.println("Client connected " + Clients.size());

}

**else**

{

Clients.remove(Clients.size()-1);

}

}

**catch** (Exception e)

{

e.printStackTrace();

}

}

}

/\*\*

\* Leert die Tabelle history und setzt Systemvariablen im DBMS

\*

\* **@param** stmt Das SQL-Statement, was mit dem DB-Server verbunden ist

\*/

**public** **static** **void** trunc\_table(java.sql.Statement stmt) **throws** SQLException

{

stmt.execute("TRUNCATE TABLE history");

stmt.execute("SET sql\_log\_bin = 0");

}

/\*\*

\* Holt den Kontostand des Accounts von der DB

\*

\* **@param** stmt Das SQL-Statement, was mit dem DB-Server verbunden ist

\* **@param** ACCID Die accid, von dem der kontostand geholt werden soll

\* **@return** Kontostand des Accounts

\* **@throws** SQLException Fehler beim Verarbeiten

\*/

**public** **static** **int** Kontostand\_TX(java.sql.Statement stmt, **int** ACCID) **throws** SQLException

{

ResultSet rs = stmt.executeQuery("select balance from accounts where accid="+ACCID+";");

rs.first();

**return** rs.getInt(1);

}

/\*\*

\* Zahlt in ein Konto ein

\*

\* **@param** stmt Das SQL-Statement, was mit dem DB-Server verbunden ist

\* **@param** ACCID Der Account, von dem der Kontostand geholt werden soll

\* **@param** TELLERID ID des Geldautomaten

\* **@param** BRANCHID ID der Zweigstelle

\* **@param** DELTA Betrag der Einzahlung

\* **@return** Kontostand des Accounts

\* **@throws** SQLException Fehler beim Verarbeiten

\*/

**public** **static** **int** Einzahlungs\_TX(java.sql.Statement stmt, **int** ACCID, **int** TELLERID, **int** BRANCHID, **int** DELTA) **throws** SQLException

{

ResultSet rs = stmt.executeQuery("SELECT Einzahlungs\_TX("+ACCID+","+TELLERID+","+BRANCHID+","+DELTA+");");

rs.first();

**return** rs.getInt(1) + DELTA;

}

/\*\*

\* Holt die Anzahl der Eintrage in history mit dem gegebene DELTA von der DB

\*

\* **@param** stmt Das SQL-Statement, was mit dem DB-Server verbunden ist

\* **@param** DELTA Das DELTA, welches in history gesucht werden soll

\* **@return** Anzahl Der gefundenen Ergebnisse

\* **@throws** SQLException Fehler beim Verarbeiten

\*/

**public** **static** **int** Analyse\_TX(java.sql.Statement stmt, **int** DELTA) **throws** SQLException

{

ResultSet rs = stmt.executeQuery("select count(accid) from benchmark.history where delta="+ DELTA +";");

rs.first();

**return** rs.getInt(1);

}

/\*\*

\* Diese Methode gibt eine gewichtet Zufallszahl zwischen 1 und 3 zurück,

\* mit einer Gewichtung von (35 : 50 : 15)

\*

\* **@return** Gewichtete Zufallszahl

\*/

**private** **static** **int** rand()

{

Random rnd = **new** Random();

**int** x = rnd.nextInt(100) + 1;

**if**(x<= 50)

**return** 1;

**if**(x<=85)

**return** 2;

**return** 3;

}

/\*\*

\* Erzeugt ein neues Object von der Connect Klasse

\*

\* **@param** args Kommandozeilenparameter

\* **@throws** SQLException Fehler beim Verarbeiten

\*/

**public** **static** **void** main(String[] args) **throws** SQLException

{

**try** {

@SuppressWarnings("unused")

Connect conc = **new** Connect();

} **catch** (ClassNotFoundException | IOException e) {

e.printStackTrace();

}

}

}