**Lab 4**

Group 1

Lab team: Ievgenii Nudga, Aryan Sultan

Instructor: Florian Ocker

HAW Hamburg Data Base

**Assignment 11: Subselects**

**A) Create SQL statements to display information about the most expensive and the cheapest articles**

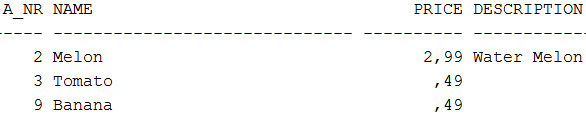
SELECT \* FROM article WHERE price **=** (SELECT MAX (price) FROM article) OR price = (SELECT MIN (price) FROM article);

or

SELECT \* FROM article WHERE price **IN** (SELECT MAX (price) FROM article) OR price IN (SELECT MIN (price) FROM article);

or

SELECT \* FROM article WHERE price **=** ( (SELECT MAX (price) FROM article) UNION (SELECT MIN (price) FROM article) );



**B) Delete Bob's father with a SQL statement that uses only 'Bob' as input**

UPDATE person SET father = null WHERE name1 = 'Bob';

**Assignment 12: Transactions**

**A)**

CREATE TABLE TAB10 (id integer CONSTRAINT pk\_tab10 PRIMARY KEY, n integer);

* The table is visible in session 2 (in our case it’s “session\_1\_1”) both before and after committing in session 1

**B)**

INSERT INTO TAB10 Values (1,1);

INSERT INTO TAB10 Values (2,2);

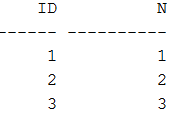
INSERT INTO TAB10 Values (3,3);

* Before commit the changes in session 1 no changes in session 2 would be visible.
* After commit the changes would be updated and can be accessible in session 2 as well.

**Output of session 2 before commit in session 1:**



**Output of session 2 after commit in session 1:**

****

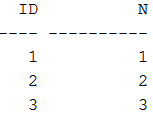
**C)**

UPDATE TAB10 SET N=33 WHERE id=3;

Before rollback in session 1:



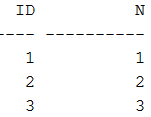
Before rollback in session 2:



After rollback in session 1:



After rollback in session 2:



Since in session 1 no commit happened => the value won’t change by update neither after rollback.

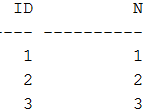
D)

UPDATE TAB10 SET N=(N\*2) WHERE id=1;

**Output in session 1:**



**Output in session 2:**

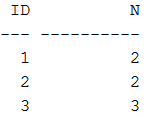
****

* In the first session no commit was made after the update and thus session 2 didn’t get newly updated data due to deadlock
* Also, if sin session 2 we try to do N\*3, then it would be same result (1) due to deadlock. To be exact, it might seem like we did 1\*3, but in reality it didn’t occur and 1 simply stayed same way without being multiplied by 3
* But after the commit in session 1 the lock will be released and in session 2 new data will be visible

Now let’s try this:

UPDATE TAB10 SET N=(N\*3) WHERE id=1;

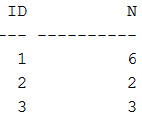
**Output in sessions 1 and 2:**

****

**Output in session 1 after commit:**

****

**Output in session 2 (this result was possible due to the commit in session 1):**

****

**Assignment 13: JDBC**

**Main**

**package** de.haw.ie4lab4;

**import** java.io.IOException;

**import** java.sql.SQLException;

**public** **class** Main {

// **TODO**: provide the correct class name of the driver!

**static** **final** String *driverName* = "oracle.jdbc.driver.OracleDriver";

// **TODO**: provide the correct JDBC-URL for the HAW database!

**static** **final** String *url* = "jdbc:oracle:thin:@ora14.informatik.haw-hamburg.de:1521:inf14";

**static** **final** String *user* = MyDBUserPassword.*user*;

**static** **final** String *password* = MyDBUserPassword.*password*;

/\*\*

\* **@param** args

\* **@throws** IOException

\*/

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

DbHandler db = **new** DbHandler();

**try** {

db.connectDB(*driverName*, *url*, *user*, *password*);

db.printOrderNumbers("Ringo");

db.printOrderNumbers("John");

db.printOrderNumbers("O'Hara");

**int** orderNumber = 5;

db.printInvoiceForOrder(orderNumber);

db.insertNewCustomer(5, "Whoever");

db.changeArticlePrice("Apple", 1.23);

}

**catch** (SQLException e)

{

// **TODO**: print stack trace!

e.printStackTrace();

// **TODO**: Print nice error message using System.err.println() and db.getSql()!

System.*err*.println(db.getSql());

}

**finally**

{

// **TODO**: close connection

db.close();

}

}

}

**DbHandler:**

**package** de.haw.ie4lab4;

**import** java.sql.Connection;

**import** java.sql.DatabaseMetaData;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** DbHandler {

/\*\*

\* Database connection

\*/

**private** Connection conn;

/\*\*

\* The current SQL statement

\*/

**private** String sql;

/\*\*

\* Getter for the current SQL statement

\*

\* **@return** the SQL statement

\*/

**public** String getSql() {

**return** sql;

}

/\*\*

\* Connect to the database.

\*

\* **@param** driverName

\* - name of JDBC driver class

\* **@param** url

\* - JDBC URL

\* **@param** user

\* - DB user name

\* **@param** password

\* - DB password

\* **@throws** SQLException

\*/

**public** **void** connectDB(String driverName, String url, String user, String password) **throws** SQLException {

System.*out*.println("Trying to connect to " + url);

// **TODO**: connect to the DB!

**try** {

Class.*forName*(driverName);

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

conn = DriverManager.*getConnection*(url, user, password);

// **TODO**: disable autoCommit!

conn.setAutoCommit(**false**);

//

// Print success message and some meta data:

//

DatabaseMetaData metaData = conn.getMetaData();

System.*out*.println("Connected to DB " + metaData.getURL() + " as user " + metaData.getUserName());

System.*out*.println(metaData.getDatabaseProductName() + " " + metaData.getDatabaseMajorVersion() + "."

+ metaData.getDatabaseMinorVersion());

}

/\*\*

\* Close the connection

\*/

**public** **void** close() {

/\*

\* **TODO**: rollback the transaction (in real life, you'd want to commit -> but

\* then you cannot call insertNewCustomer() twice.)

\*/

// **TODO**: close the connection (if it has been initialized)

**try** {

conn.rollback();

conn.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Print the list of order numbers for the given customer

\*

\* **@param** customer

\* - Name of customer

\* **@throws** SQLException

\*/

**public** **void** printOrderNumbers(String customer) **throws** SQLException {

System.*out*.println("\n" + customer + "'s orders:");

// **TODO**: SQL see assignment 10b-10

Statement st = conn.createStatement();

sql= "SELECT o\_nr FROM customer c, orders o WHERE c.name = ? AND c.c\_id = o.c\_id";

PreparedStatement pst =conn.prepareStatement(sql);

pst.setString(1, customer);

ResultSet cursor = pst.executeQuery();

**while** (cursor.next()) {

**int** i = cursor.getInt(1);

System.*out*.print(i+" ");

}

cursor.close();

st.close();

}

/\*\*

\* Print an invoice for the given order. The invoice shall contain every

\* single order item and the total price.

\*

\* **@param** orderNumber

\* - value for o\_nr

\* **@throws** SQLException

\*/

**public** **void** printInvoiceForOrder(**int** orderNumber) **throws** SQLException {

System.*out*.println("\nInvoice for order number " + orderNumber);

// Optional: You could print customer information here!

/\*

\* **TODO**: For every order item, print the article name, the article's price

\* per unit, the quantity, and the price of the order item. SQL see

\* assignment 10b-14

\*/

Statement st = conn.createStatement();

sql= "SELECT name,price , quantity , price \* quantity FROM orders o, order\_item ort INNER JOIN article art ON ort.a\_nr=art.a\_nr WHERE o.o\_nr=? and ort.o\_nr=? ORDER BY name ASC";

PreparedStatement pst =conn.prepareStatement(sql);

pst.setInt(1, orderNumber);

pst.setInt(2, orderNumber);

**int** n = pst.executeUpdate();

ResultSet cursor = pst.executeQuery();

**double** d3=0;

**while** (cursor.next()) {

// position in cursor starts at 1!

String s1 = cursor.getString(1);

**double** d1 = cursor.getDouble(2);

**int** i1 = cursor.getInt(3);

**double** d2 = cursor.getDouble(4);

d3+=d2;

//int i2 = cursor.getInt(2);

System.*out*.println(s1+" "+d1+" "+i1+" "+d2+" ");

//System.out.println(i2);

}

cursor.close();

st.close();

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

/\*

\* **TODO**: Print the total price of the order. You can calculate the sum via

\* SQL (see assignment 10b-15), or in Java.

\*/

System.*out*.println("Order's total price: "+d3);

//sql = "";

}

/\*\*

\* Insert a new customer

\*

\* **@param** id

\* - customer ID

\* **@param** name

\* - customer name

\* **@throws** SQLException

\*/

**public** **void** insertNewCustomer(**int** id, String name) **throws** SQLException {

System.*out*.println("Trying to insert new customer. id=" + id + ", name=" + name);

// **TODO**: insert a new customer with the given values

Statement st = conn.createStatement();

sql = "INSERT INTO customer VALUES (?, ?)";

PreparedStatement pst =conn.prepareStatement(sql);

pst.setInt(1, id);

pst.setString(2, name);

pst.execute();

st.close();

}

/\*\*

\* Change the article's price

\*

\* **@param** articleName

\* - identifies the article

\* **@param** price

\* - the new price

\* **@throws** SQLException

\*/

**public** **void** changeArticlePrice(String articleName, **double** price) **throws** SQLException {

System.*out*.println("Trying to set the price of " + articleName + " to " + price);

//sql = "";

**int** n = 0;

// **TODO**: change the article's price

sql = "UPDATE article SET price = ? WHERE name= ?";

PreparedStatement pst =

conn.prepareStatement(sql);

pst.setDouble(1, price);

pst.setString(2, articleName);

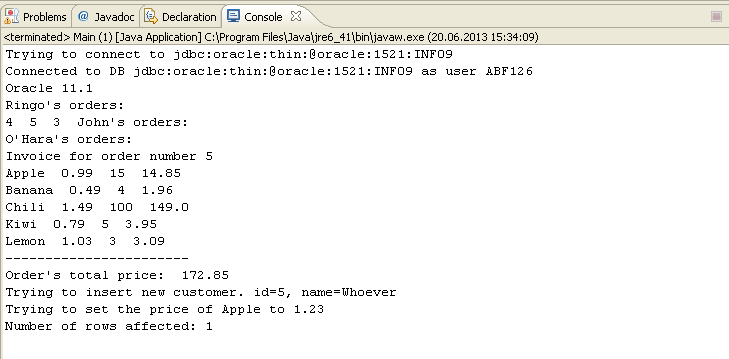
n = pst.executeUpdate();

System.*out*.println("Number of rows affected: " + n);

}

}

**See next page for the output we need (though, it doesn’t seem accurate – near Ringo’s order there ae 4 5 3, and near other orders there’s nothing):**



**Output I get: (the problem seems to be in “Trying to insert the price of…”)**

