INF.01014UF DATABASES - Report

Simone Franza 01530693
Lukas Meer 01430417

Table of content

1. Database schema

Domains:	**CId**, integer	Id-number of the customer
	CName, string	Name of the customer
	CCity, string	City where the customer lives
	CAge, integer	Age of the customer
	CNumber, string	Telephone number of the customer
	BId, integer	Id-number of kind of beer
	BName, string	Name of a beer
	BPrice, integer	Price of a beer
	TId, integer	Id-number of the transaction
	TDate, date	Date of the purchase
	TQnt, integer	Quantity of a kind of beer bought by a customer
	RRating, integer	Rating that a customer gave to a kind of beer

- **Relation: customer** (CId, CName, CCity, CAge, CNumber)
- **Relation: beer** (BId, BName, BPrice)
- **Relation: transaction** (TId, CId, BId, TDate, TQnt)
- **Relation: rating** (CId, BId, RRating)

2. Functional dependencies

Relation: customer (CId, CName, CCity, CAge, CNumber)
CId → CName
CId → CCity
CId → CAge
CId → CNumber
Relation: beer (BId, BName, BPrice)
BId → BName
BId → BPrice
Relation: transaction (TId, CId, BId, TDate, TQnt)
TId → CId
TId → BId
TId → TDate
TId → TQnt
Relation: rating (CId, BId, RRating)
(CId, BId)→ RRating

Definition of the 3rd normal form:
“Third normal form (3NF) is the third step in normalizing a database and it builds on the first and second normal forms, 1NF and 2NF.
3NF states that all column reference in referenced data that are not dependent on the primary key should be removed. Another way of putting this is that only foreign key columns should be used to reference another table, and no other columns from the parent table should exist in the referenced table.”

Source: https://www.techopedia.com/definition/22561/third-normal-form-3nf (visited: 05/06/2018)

As shown at the beginning of Chapter 2 all our domains in their relations only depend on their respective primary key, i.e. they don't depend on any other domains.

In the relation “customer” all columns depend only of the primary key (CId).

For the relation “beer” we assumed, that the entries of the column BName identify a single model of beer (for example Gösser Naturradler). This model can have different sizes, therefore BPrice doesn't depend of BName.

For the relation “transaction” we assigned to every transaction an Id. Every customer can buy multiple products a day and in different quantities so every column only depends of TId. The column TQnt shows the number of bottles of a certain beer model (BId) that a costumer bought.

For the relation “rating” we assumed, that every customer can assign only one rating to a certain model of beer (BId). Therefore RRating cannot depend only of CId or of BId.

Additionally all other requirements are also fulfilled (2nd normal form), so our database is in the 3rd normal form.

3. Current state of the database

customer				
CId	**CName**	**CCity**	**CAge**	**CNumber**
1	Mueller	Graz	20	00436601111222
2	Franza	Muenchen	30	00493149264872
3	Maar	Graz	35	00436602222333
4	Rossi	Rom	50	00393486805555
5	Pranger	Innsbruck	25	00436504262333
6	Hager	Wien	45	00436958728333
7	Muster	Heidelberg	25	00493149226872
8	Ferrari	Florenz	35	00393486195255
9	Heine	Graz	27	00436602182433
10	Musk	Innsbruck	97	00436605262339

beer		
BId	**BName**	**BPrice**
1	Goesser Naturradler	5
2	Goesser Naturradler	4
3	Stiegl Helles	3
4	Zillertaler Weizen	4
5	Murauer Helles	3

transaction				
TId	**CId**	**BId**	**TDate**	**TQnt**
1	1	1	2018-04-10	5
2	1	2	2018-04-10	3
3	2	1	2018-04-15	6
4	3	2	2018-04-15	1
5	6	5	2018-04-20	10
6	4	4	2018-04-25	6
7	8	5	2018-05-10	3
8	8	3	2018-05-17	7
9	10	4	2018-05-30	24

transaction				
TId	**CId**	**BId**	**TDate**	**TQnt**
10	5	2	2018-06-01	4
11	3	2	2018-06-03	9
12	6	3	2018-06-03	2

rating		
CId	**BId**	**RRating**
1	1	3
1	3	2
2	1	5
3	2	2
5	1	4
5	2	3
6	5	0
10	3	2

4. Database queries in terms of the Relational Algebra

Get the names of beer and ratings for consumer, which live in Graz

```
Select customer Where CCity = 'Graz' Giving A;
Join A And rating over CId Giving B;
Join B And beer over BId Giving C;
Project C Over BName, RRating Giving RESULT;
```

Get names of those customer, who bought “Stiegl Helles” or live in “Innsbruck”

```
Select customer Where CCity = 'Innsbruck' Giving A;
Project A Over CName Giving X;
Select beer Where BName = 'Stiegl Helles' Giving B;
Join B And transaction Over BId Giving C;
Join C And customer Over CId Giving D;
Project D Over CName Giving Y;
X Union Y Giving RESULT;
```

Get names of those customers, who bought both type of “Goesser Naturradler”

```
Select beer Where BName = 'Goesser Naturradler' Giving A;
Project A Over BId Giving X;
Project transaction Over CId, BId Giving Y;
Divide Y By X Giving C;
Join C And customer Over CId Giving D;
Project D Over CName Giving RESULT;
```

5. Database queries in terms of the Relational Calculus

Get the model of beer which sold the most on a single transaction

B -> beer
T1 -> transaction
T2 -> transaction

(B.BName): ∃T1 ∀T2 (B.BId = T1.BId & T1.TQnt >= T2.TQnt)

———

Get the names of the customers who bought “Murauer Helles”

C -> customer
T -> transaction
B -> beer

(C.CName): ∃T ∃B (C.CId = T.CId & T.BId = B.BId & B.BName = 'Murauer Helles')

———

Get the ratings of the model of beer which sold the least in a single transaction

R -> rating
T1 -> transaction
T2 -> transaction
B -> beer

(R.RRating): ∃T1 ∀T2 ∃B (R.BId = B.BId & B.Bid = T1.BId & T1.TQnt <= T2.TQnt)

6. Database queries in terms of the SQL

Get the names and the telephone numbers from all the customers who are older than 30 and don't live in “Innsbruck” and bought “Stiegl Helles”. Sort the results by name

```
SELECT CName, CNumber FROM
customer WHERE (CId IN
        (SELECT CId FROM transaction
        WHERE BId IN
                (SELECT BId FROM beer
                WHERE BName = 'Stiegl Helles')))
AND (CAge >= 30)
AND NOT (CCity = 'Innsbruck')
ORDER BY CName ASC;
```

———

Get the city of the customers who bought more than 10 products over all transactions.

```
SELECT CCity FROM
customer WHERE (CId IN
        (SELECT CId FROM transaction
        GROUP BY CId HAVING SUM(TQnt) > 10))
ORDER BY CCity DESC;
```

———

Get the name and ids of the beers of which were bought more than 15 units or are called “Murauer Helles”

```
SELECT BId, BName FROM
beer WHERE (BId IN
        (SELECT BId FROM transaction
        GROUP BY BId HAVING SUM(TQnt) > 15))
OR (BName = 'Murauer Helles')
ORDER BY BId ASC;
```

7. Database queries in terms of the SQL without nested SQL blocks

Get the names and the telephone numbers from all the customers who are older than 30 and don't live in “Innsbruck” and bought “Stiegl Helles”. Sort the results by name

```
SELECT CName, CNumber FROM customer, transaction, beer
WHERE customer.CId = transaction.CId
AND transaction.BId = beer.BId
AND BName = 'Stiegl Helles'
AND CAge >= 30
AND NOT CCity = 'Innsbruck'
ORDER BY CName ASC;
```

Get the city of the customers who bought more than 10 products over all transactions.

```
SELECT CCity FROM customer, transaction
WHERE customer.CId = transaction.CId
GROUP BY transaction.CId HAVING SUM(TQnt) > 10
ORDER BY CCity DESC;
```

Get the name and ids of the beers of which were bought more than 15 units or are called “Murauer Helles”

```
SELECT beer.BId, BName FROM beer, transaction
WHERE BName = 'Murauer Helles'
OR beer.BId = transaction.BId
GROUP BY beer.BId HAVING SUM(TQnt) > 15
ORDER BY beer.BId ASC;
```

8. Practical implementation of the database with SQL

```
CREATE DATABASE 01530693_beer;
USE 01530693_beer

# create all relations

CREATE TABLE customer (
        CId INTEGER NOT NULL,
        CName VARCHAR(30) NOT NULL,
        CCity VARCHAR(30) NOT NULL,
        CAge INTEGER NOT NULL,
        CNumber VARCHAR(20) NOT NULL,
PRIMARY KEY (CId));

CREATE TABLE beer (
        BId INTEGER NOT NULL,
        BName VARCHAR(40) NOT NULL,
        BPrice INTEGER NOT NULL,
PRIMARY KEY (BId));

CREATE TABLE transaction (
        TId INTEGER NOT NULL,
        CId INTEGER NOT NULL,
        BId INTEGER NOT NULL,
        TDate DATE NOT NULL,
        TQnt INTEGER NOT NULL,
PRIMARY KEY (TId),
FOREIGN KEY (CId) REFERENCES customer(CId) ON DELETE CASCADE,
FOREIGN KEY (BId) REFERENCES beer(BId) ON DELETE CASCADE);

CREATE TABLE rating (
        CId INTEGER NOT NULL,
        BId INTEGER NOT NULL,
        RRating INTEGER NOT NULL,
PRIMARY KEY (CId, BId),
FOREIGN KEY (CId) REFERENCES customer(CId) ON DELETE CASCADE,
FOREIGN KEY (BId) REFERENCES beer(BId) ON DELETE CASCADE);

# insert content into the relations:

INSERT INTO customer VALUES
        (1, 'Mueller', 'Graz', 20, '00436601111222'),
        (2, 'Franza', 'Muenchen', 30, '00493149264872'),
        (3, 'Maar', 'Graz', 35, '00436602222333'),
        (4, 'Rossi', 'Rom', 50, '00393486805555'),
        (5, 'Pranger', 'Innsbruck', 25, '00436504262333'),
        (6, 'Hager', 'Wien', 45, '00436958728333'),
        (7, 'Muster', 'Heidelberg', 25, '00493149226872'),
        (8, 'Ferrari', 'Florenz', 35, '00393486195255'),
        (9, 'Heine', 'Graz', 27, '00436602182433'),
        (10, 'Musk', 'Innsbruck', 97, '00436605262339');

INSERT INTO beer VALUES
        (1, 'Goesser Naturradler', 5),
        (2, 'Goesser Naturradler', 4),
        (3, 'Stiegl Helles', 3),
        (4, 'Zillertaler Weizen', 4),
        (5, 'Murauer Helles', 3);
```

```
INSERT INTO transaction VALUES
        (1, 1, 1, '2018-04-10', 5),
        (2, 1, 2, '2018-04-10', 3),
        (3, 2, 1, '2018-04-15', 6),
        (4, 3, 2, '2018-04-15', 1),
        (5, 6, 5, '2018-04-20', 10),
        (6, 4, 4, '2018-04-25', 6),
        (7, 8, 5, '2018-05-10', 3),
        (8, 8, 3, '2018-05-17', 7),
        (9, 10, 4, '2018-05-30', 24),
        (10, 5, 2, '2018-06-01', 4),
        (11, 3, 2, '2018-06-03', 9),
        (12, 6, 3, '2018-06-03', 2);

INSERT INTO rating VALUES
        (1, 1, 3),
        (1, 3, 2),
        (2, 1, 5),
        (3, 2, 2),
        (5, 1, 4),
        (5, 2, 3),
        (6, 5, 0),
        (10, 3, 2);

# create user for the servlets

CREATE USER 'student'@'localhost' IDENTIFIED BY 'student';
GRANT ALL PRIVILEGES ON 01530693_beer.* TO 'student'@'localhost' WITH GRANT OPTION;

# print the content of every table (to test if everything is right)

SELECT * FROM customer;
SELECT * FROM beer;
SELECT * FROM transaction;
SELECT * FROM rating;

# now start the queries from chapters 6 and 7
```

9. Servlets (Database Modification)

This servlet contains all HTML-Forms, which supply the following servlets with the data they require. It contains 5 HTML-Forms, for every servlet one.

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.io.PrintWriter;
/**
 * Servlet implementation class Start
 */
@WebServlet("/Start")
public class Start extends HttpServlet {
private static final long serialVersionUID = 1L;

/**
* Default constructor.
*/
public Start() {
// TODO Auto-generated constructor stub
}

/**
* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
*/
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

response.setContentType("text/html");
PrintWriter writer = response.getWriter();

writer.println("<html>");
writer.println("<head><title>Overview Servlet</title></head>");
writer.println("<body>");
writer.println("<form method='get' action='InsertRating'>");
writer.println("<h2>Insert a new rating:</h2>");
writer.println("CustomerId:<input type='text' name='customerId'/>");
writer.println("BeerId:<input type='text' name='beerId'/>");
writer.println("Rating:<input type='text' name='rRating'/>");
writer.println("<input type='submit' value='Insert Rating'/>");
writer.println("</form>");
writer.println("<form method='get' action='DeleteCustomer'>");
writer.println("<h2>Delete a customer and all the related data:</h2>");
writer.println("CustomerId:<input type='text' name='customerId'/>");
writer.println("<input type='submit' value='Delete this customer'/>");
writer.println("</form>");
```

```
writer.println("<form method='get' action='GetCustomerProduct'>");
writer.println("<h2>Get all the customer who bought at least the specified
number of products over all their transactions:</h2>");
writer.println("Amount of products:<input type='text' name='nProduct'/>");
writer.println("<input type='submit' value='Get customers'/>");
writer.println("</form>");
writer.println("<form method='get' action='GetRatingCity'>");
writer.println("<h2>Get the average rating given by the costumer living in the
specified city:</h2>");
writer.println("City:<input type='text' name='city'/>");
writer.println("<input type='submit' value='Get average ratings'/>");
writer.println("</form>");
writer.println("<form method='get' action='GetPhoneNumber'>");
writer.println("<h2>Get the telephone numbers of those customer who bought the
specified beer:</h2>");
writer.println("BeerName:<input type='text' name='beerName'/>");
writer.println("<input type='submit' value='Get telephone numbers'/>");
writer.println("</form>");
writer.println("</body>");
writer.println("</html>");
writer.close();

}
}
```

9.1. Insert a Rating into the database

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class InsertRating
 */
@WebServlet("/InsertRating")
public class InsertRating extends HttpServlet {
private static final long serialVersionUID = 1L;

/**
* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
*/
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub

response.setContentType("text/html");
PrintWriter writer = response.getWriter();
```

```
writer.println("<html>");
writer.println("<head><title>Insert Rating</title></head>");
writer.println("<body>");
writer.println("<h1>Insert Rating!</h1>");


// Get form data and check if text is empty or not
String CIdStr = request.getParameter("customerId");
String BIdStr = request.getParameter("beerId");
String RRatingStr = request.getParameter("rRating");

if((CIdStr == null) || (CIdStr.length() == 0)) {
printMsg("Cannot insert a rating with no customer Id!", writer, request);
return;
}

if((BIdStr == null) || (BIdStr.length() == 0)) {
printMsg("Cannot insert a rating with no beer Id!", writer, request);
return;
}

if((RRatingStr == null) || (RRatingStr.length() == 0)) {
printMsg("Cannot insert a rating with no rating!", writer, request);
return;
}

// Parse IDs and rating to integer
int CId, BId, RRating;

try {
CId = Integer.parseInt(CIdStr);
}
catch(NumberFormatException exc) {
exc.printStackTrace();
printMsg("Cannot insert a rating with an invalid customer Id!", writer, request);
return;
}

try {
BId = Integer.parseInt(BIdStr);
}
catch(NumberFormatException exc) {
exc.printStackTrace();
printMsg("Cannot insert a rating with an invalid beer Id!", writer, request);
return;
}

try {
RRating = Integer.parseInt(RRatingStr);
}
catch(NumberFormatException exc) {
exc.printStackTrace();
printMsg("Cannot insert a rating with an invalid rating!", writer, request);
return;
}

// Check if IDs are positive and if rating is between 0 and 5
if(CId<0) {
printMsg("Cannot insert a rating with negative customer Id!", writer, request);
```

```
return;
}

if(BId<0) {
printMsg("Cannot insert a rating with negative beer Id!", writer, request);
return;
}

if((RRating < 0) || (RRating > 5)) {
printMsg("The rating cannot be negative or greater then 5!", writer, request);
return;
}

// Connect to the database
try {
Class.forName("com.mysql.jdbc.Driver");
}
catch(ClassNotFoundException exc) {
exc.printStackTrace();
printMsg("Cannot insert the rating : no JDBC driver found!", writer, request);
return;
}

try {
Connection connection_;
connection_ = DriverManager.getConnection("jdbc:mysql://localhost/01530693_beer",
"student", "student");
Statement statement = connection_.createStatement(); // Referential integrity gets
checked here!!

ResultSet resultC = statement.executeQuery("SELECT * FROM customer WHERE CId =" + CId);

if(!resultC.next()) {
printMsg("Cannot insert the rating: no such customer!", writer, request);
return;
}

ResultSet resultB = statement.executeQuery("SELECT * FROM beer WHERE BId =" + BId);

if(!resultB.next()) {
printMsg("Cannot insert the rating: no such beer!", writer, request);
return;
}

String insertSqlStmt = "INSERT INTO rating VALUES (" + CId + "," + BId + "," + RRating +
");";
statement.executeUpdate(insertSqlStmt);

printMsg("Transaction successfully inserted!", writer, request);
}
catch(SQLException exc) {
exc.printStackTrace();
printMsg("Cannot insert rating: database error!", writer, request);
}

writer.println("</body>");
writer.println("</html>");
writer.close();
}
private void printMsg(String msg, PrintWriter writer, HttpServletRequest request) {
```

```
writer.write("<h3>" + msg + "</h3>\n");
writer.write("<a href = \"" + request.getHeader("Referer") + "\">Back</a>\n");
writer.write("</body>");
writer.write("</html>");
}
}
```

9.2. Delete a customer from the database

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class DeleteCustomer
 */
@WebServlet("/DeleteCustomer")
public class DeleteCustomer extends HttpServlet {
private static final long serialVersionUID = 1L;

/**
* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
*/
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub
response.setContentType("text/html");
PrintWriter writer = response.getWriter();

writer.println("<html>");
writer.println("<head><title>Delete Customer</title></head>");
writer.println("<body>");
writer.println("<h1>Delete Customer and all related data!</h1>");

// Get form data and check if text is empty or not
String CIdStr = request.getParameter("customerId");

if((CIdStr == null) || (CIdStr.length() == 0)) {
printMsg("Cannot delete a user with no user Id!", writer, request);
return;
}

// Parse IDs and rating to integer
int CId;

try {
CId = Integer.parseInt(CIdStr);
}
catch(NumberFormatException exc) {
exc.printStackTrace();
printMsg("Cannot delete a user with an invalid user Id!", writer, request);
```

```
return;
}

// Check if ID is positive
if(CId<0) {
printMsg("Cannot delete a user with a negative user Id!", writer, request);
return;
}

// Connect to the database
try {
Class.forName("com.mysql.jdbc.Driver");
}
catch(ClassNotFoundException exc) {
exc.printStackTrace();
printMsg("Cannot delete the user: no JDBC driver found!", writer, request);
return;
}

try {
Connection connection_;
connection_ = DriverManager.getConnection("jdbc:mysql://localhost/01530693_beer",
"student", "student");
Statement statement = connection_.createStatement(); // Referential integrity gets
checked here!!

ResultSet result = statement.executeQuery("SELECT * FROM customer WHERE CId =" + CId);

if(!result.next()) {
printMsg("Cannot delete a customer which does not exist!", writer, request);
return;
}

String deleteSqlStmt = "DELETE FROM customer WHERE CId = " + CId;
statement.executeUpdate(deleteSqlStmt);

printMsg("Customer and all related data successfully deleted!", writer, request);
}
catch(SQLException exc) {
exc.printStackTrace();
printMsg("Cannot delete customer: database error!", writer, request);
}


writer.println("</body>");
writer.println("</html>");
writer.close();
}
private void printMsg(String msg, PrintWriter writer, HttpServletRequest request) {
writer.write("<h3>" + msg + "</h3>\n");
writer.write("<a href = \"" + request.getHeader("Referer") + "\">Back</a>\n");
writer.write("</body>");
writer.write("</html>");

}
}
```

10. Servlets (Database Queries)

10.1. Get Customer

This query gets all the customer who bought at least the specified number of products over all their transactions.

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class GetCustomerProduct
 */
@WebServlet("/GetCustomerProduct")
public class GetCustomerProduct extends HttpServlet {
private static final long serialVersionUID = 1L;

/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub
response.setContentType("text/html");
PrintWriter writer = response.getWriter();

writer.println("<html>");
writer.println("<head><title>Get Customer</title>");
writer.println("<style>table, th, td {text-align:center; border: 1px solid black; border-
collapse: collapse; padding: 5px;}</style></head>");
writer.println("<body>");
writer.println("<h1>Get all the customer who bought at least the specified number of
products over all their transactions:</h1>");

// Get form data and check if text is empty or not
String NProdStr = request.getParameter("nProduct");

if((NProdStr == null) || (NProdStr.length() == 0)) {
printMsg("You need to enter a number of products!", writer, request);
return;
}

// Parse number of products to integer
int NProd;

try {
NProd = Integer.parseInt(NProdStr);
}
catch(NumberFormatException exc) {
exc.printStackTrace();
```

```
printMsg("Cannot execute a query with an invalid number of product!", writer, request);
return;
}

// Check if number of product is positive
if(NProd<0) {
printMsg("The number of products needs to be positive!", writer, request);
return;
}

// Connect to the database
try {
Class.forName("com.mysql.jdbc.Driver");
}
catch(ClassNotFoundException exc) {
exc.printStackTrace();
printMsg("Cannot execute the query: no JDBC driver found!", writer, request);
return;
}

try {
Connection connection_;
connection_ = DriverManager.getConnection("jdbc:mysql://localhost/01530693_beer",
"student", "student");
Statement statement = connection_.createStatement();

String myQuery = "SELECT customer.CId, CName, CCity, CAge, CNumber, SUM(TQnt) as NProduct
FROM customer,transaction WHERE customer.CId = transaction.CId GROUP BY transaction.TId
HAVING SUM(TQnt)>" + NProd + " ORDER BY customer.CId ASC"; // Select from multiple
relations, GROUP BY and HAVING are in this query

ResultSet result = statement.executeQuery(myQuery);

boolean exist = false;
writer.println("<table>");

if (result.next()) {
exist = true;
writer.println("<tr>");
writer.println("<th>CId</th>");
writer.println("<th>CName</th>");
writer.println("<th>CCity</th>");
writer.println("<th>CAge</th>");
writer.println("<th>CNumber</th>");
writer.println("<th>NProduct</th>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>" + result.getInt("CId") + "</td>");
writer.println("<td>" + result.getString("CName") + "</td>");
writer.println("<td>" + result.getString("CCity") + "</td>");
writer.println("<td>" + result.getInt("CAge") + "</td>");
writer.println("<td>" + result.getString("CNumber") + "</td>");
writer.println("<td>" + result.getInt("NProduct") + "</td>");
writer.println("</tr>");
}

while (result.next())
{
writer.println("<tr>");
writer.println("<td>" + result.getInt("CId") + "</td>");
```

```
writer.println("<td>" + result.getString("CName") + "</td>");
writer.println("<td>" + result.getString("CCity") + "</td>");
writer.println("<td>" + result.getInt("CAge") + "</td>");
writer.println("<td>" + result.getString("CNumber") + "</td>");
writer.println("<td>" + result.getInt("NProduct") + "</td>");
writer.println("</tr>");
}
writer.println("</table>");
if (exist == false)
{
printMsg("No such customer found", writer, request);
}
else
{
writer.println("<br/><a href = \"" + request.getHeader("Referer") + "\">Back</a>\n");
}
}
catch(SQLException exc) {
exc.printStackTrace();
printMsg("Cannot search customer: database error!", writer, request);
}

writer.println("</body>");
writer.println("</html>");
writer.close();
}
private void printMsg(String msg, PrintWriter writer, HttpServletRequest request) {
writer.write("<h3>" + msg + "</h3>\n");
writer.write("<a href = \"" + request.getHeader("Referer") + "\">Back</a>\n");
writer.write("</body>");
writer.write("</html>");

}
}
```

10.2. Get Average Rating

This query gets the average rating given by the costumer living in the specified city.

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class GetRatingCity
 */
@WebServlet("/GetRatingCity")
public class GetRatingCity extends HttpServlet {
private static final long serialVersionUID = 1L;
/**
* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
*/
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub
response.setContentType("text/html");
PrintWriter writer = response.getWriter();

writer.println("<html>");
writer.println("<head><title>Get Ratings</title>");
writer.println("<style>table, th, td {text-align:center; border: 1px solid black; border-
collapse: collapse; padding: 5px;}</style></head>");
writer.println("<body>");
writer.println("<h1>Get the average rating given by the costumer living in the specified
city:</h1>");

// Get form data and check if text is empty or not
String CCity = request.getParameter("city");

if((CCity == null) || (CCity.length() == 0)) {
printMsg("You need to enter a city!", writer, request);
return;
}

// Connect to the database
try {
Class.forName("com.mysql.jdbc.Driver");
}
catch(ClassNotFoundException exc) {
exc.printStackTrace();
printMsg("Cannot execute the query: no JDBC driver found!", writer, request);
return;
}

try {
Connection connection_;
connection_ = DriverManager.getConnection("jdbc:mysql://localhost/01530693_beer",
"student", "student");

Statement statement = connection_.createStatement();

// Check if city is in the database
ResultSet testCity = statement.executeQuery("SELECT * FROM customer WHERE CCity ='" +
CCity + "'");

if(!testCity.next()) {
printMsg("Cannot execute with the query: the specified city is not available in the
database!", writer, request);
return;
}

String myQuery = "SELECT customer.CId, CName, CCity, ROUND(AVG(RRating),2) as
AverageRating FROM customer, rating WHERE customer.CId = rating.CId AND CCity = '" +
CCity + "' GROUP BY rating.CId ORDER BY customer.CId ASC";

ResultSet result = statement.executeQuery(myQuery);

boolean exist = false;
writer.println("<table>");

if (result.next()) {
exist = true;
```

```
writer.println("<tr>");
writer.println("<th>CId</th>");
writer.println("<th>CName</th>");
writer.println("<th>CCity</th>");
writer.println("<th>Average Rating</th>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>" + result.getInt("CId") + "</td>");
writer.println("<td>" + result.getString("CName") + "</td>");
writer.println("<td>" + result.getString("CCity") + "</td>");
writer.println("<td>" + result.getFloat("AverageRating") + "</td>");
writer.println("</tr>");
}

while (result.next())
{
writer.println("<tr>");
writer.println("<td>" + result.getInt("CId") + "</td>");
writer.println("<td>" + result.getString("CName") + "</td>");
writer.println("<td>" + result.getString("CCity") + "</td>");
writer.println("<td>" + result.getInt("AverageRating") + "</td>");
writer.println("</tr>");
}
writer.println("</table>");
if (exist == false)
{
printMsg("The customers living in the specified city didn't give any rating!", writer,
request);
}
else
{
writer.println("<br/><a href = \"" + request.getHeader("Referer") + "\">Back</a>\n");
}
}
catch(SQLException exc) {
exc.printStackTrace();
printMsg("Cannot search rating: database error!", writer, request);
}

writer.println("</body>");
writer.println("</html>");
writer.close();
}
private void printMsg(String msg, PrintWriter writer, HttpServletRequest request) {
writer.write("<h3>" + msg + "</h3>\n");
writer.write("<a href = \"" + request.getHeader("Referer") + "\">Back</a>\n");
writer.write("</body>");
writer.write("</html>");

}
}
```

10.3. Get Telephone Number

This query gets the telephone numbers of those customer who bought the specified beer.

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.sql.Statement;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class GetPhoneNumber
 */
@WebServlet("/GetPhoneNumber")
public class GetPhoneNumber extends HttpServlet {
private static final long serialVersionUID = 1L;

/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub
response.setContentType("text/html");
PrintWriter writer = response.getWriter();

writer.println("<html>");
writer.println("<head><title>Get Numbers</title>");
writer.println("<style>table, th, td {text-align:center; border: 1px solid black; border-
collapse: collapse; padding: 5px;}</style></head>");
writer.println("<body>");
writer.println("<h1>Get the telephone numbers of those customer who bought the specified
beer:</h1>");

// Get form data and check if text is empty or not
String BName = request.getParameter("beerName");

if((BName == null) || (BName.length() == 0)) {
printMsg("You need to enter a beer name!", writer, request);
return;
}

// Connect to the database
try {
Class.forName("com.mysql.jdbc.Driver");
}
catch(ClassNotFoundException exc) {
exc.printStackTrace();
printMsg("Cannot execute the query: no JDBC driver found!", writer, request);
return;
}

try {
Connection connection_;
connection_ = DriverManager.getConnection("jdbc:mysql://localhost/01530693_beer",
"student", "student");

Statement statement = connection_.createStatement();

// Check if city is in the database
```

```
ResultSet testCity = statement.executeQuery("SELECT * FROM beer WHERE BName ='" + BName +
"'");

if(!testCity.next()) {
printMsg("Cannot execute with the query: the specified beer name is not available in the
database!", writer, request);
return;
}

String myQuery = "SELECT DISTINCT customer.CId, CName, CNumber FROM customer,
transaction, beer WHERE customer.CId = transaction.CId AND beer.BId = transaction.BId AND
BName = '" + BName + "' ORDER BY customer.CId ASC";

ResultSet result = statement.executeQuery(myQuery);

boolean exist = false;
writer.println("<table>");

if (result.next()) {
exist = true;
writer.println("<tr>");
writer.println("<th>CId</th>");
writer.println("<th>CName</th>");
writer.println("<th>CNumber</th>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>" + result.getInt("CId") + "</td>");
writer.println("<td>" + result.getString("CName") + "</td>");
writer.println("<td>" + result.getString("CNumber") + "</td>");
writer.println("</tr>");
}

while (result.next())
{
writer.println("<tr>");
writer.println("<td>" + result.getInt("CId") + "</td>");
writer.println("<td>" + result.getString("CName") + "</td>");
writer.println("<td>" + result.getString("CNumber") + "</td>");
writer.println("</tr>");
}
writer.println("</table>");
if (exist == false)
{
printMsg("The specified beer wasn't bought by any customer yet!", writer, request);
}
else
{
writer.println("<br/><a href = \"" + request.getHeader("Referer") + "\">Back</a>\n");
}
}
catch(SQLException exc) {
exc.printStackTrace();
printMsg("Cannot search beer: database error!", writer, request);
}

writer.println("</body>");
writer.println("</html>");
writer.close();
}
private void printMsg(String msg, PrintWriter writer, HttpServletRequest request) {
```

```
writer.write("<h3>" + msg + "</h3>\n");
writer.write("<a href = \"" + request.getHeader("Referer") + "\">Back</a>\n");
writer.write("</body>");
writer.write("</html>");

}
}
```