Day 1 exercises (Stored procedures, indexing and prepared statements)

These exercises train your skills working with non-functional requirements. Focus is on performance and maintainability, i.e. how to reduce response time by applying indexing and simplify sql queries by sending parameters to a stored procedure.

In some systems the only way you will be allowed to access data is by running a stored procedure, that was created for you by the database manager.

Exercise 1 – Stored Procedure

Make a stored procedure for the parts table from the mail order database (the one you worked with in study point assignment last week).

The result of the stored procedure is based on a condition so you must use the MySQL IF statement.

a) Create the stored procedure like this in MySQL Workbench:

```
DELIMITER //
 CREATE PROCEDURE GetPartLevel(in  partNumber int(11),
    out partLevel  varchar(10))
   BEGIN
   DECLARE priceLevel double;

    SELECT price INTO priceLevel
    FROM parts
    WHERE pno = partNumber;

    IF priceLevel > 20 THEN
        SET partLevel = 'HIGH';
    ELSEIF (priceLevel <= 20 AND priceLevel >= 15) THEN
        SET partLevel = 'MEDIUM';
    ELSEIF priceLevel < 15 THEN
        SET partLevel = 'LOW';
    END IF;
   END //
 DELIMITER ;
```

b) Execute the stored procedure like this in MySQL Workbench:

```
call GetPartLevel(10800, @level);
select @level;
```

c) Execute the stored procedure from Java code, for instance like this:

```
public String getPriceLevel(int pno) {
   String level = null;
   CallableStatement stmt = null;
   try {
     stmt = con.prepareCall("{call GetPartLevel(?,?)}");
     stmt.setInt(1, pno);
     stmt.registerOutParameter(2, java.sql.Types.VARCHAR);
     stmt.execute();
     level = stmt.getString(2);
    }
     catch (SQLException ignore) {
     }
        return level;
    }
```

d) Unit test the mapper method. You must test all three levels (HIGH, MEDIUM, LOW) in each their JUnit test method.

Exercise 2 – Stored Procedures

Make a stored procedure for the parts table from the mail order database what returns the number of parts for a given order level.

Example:

For the following parts table data, the procedure should return **5** for order level **20**.

pno	pname	qoh	price	olevel
10506	Land Before Time I	200	19.99	20
10507	Land Before Time II	156	19.99	20
10508	Land Before Time III	190	19.99	20
10509	Land Before Time IV	60	19.99	20
10601	Sleeping Beauty	300	24.99	20
10701	When Harry Met Sally	120	19.99	30
10800	Dirty Harry	140	14.99	30
10900	Dr. Zhivago	100	24.99	30

Exercises 3 - Indexing

It is possible to increase response time by putting an index on one or more columns. Primary and foreign keys are automatically indexed in MYSQL InnoDB. You might check this via the MySQL Workbench Table Inspector on the Indexes tab.

If users often search products by product name it could be a good idea to index that column.

a) In your mail order database, retrieve one or more products from the parts table with a SQL statement like this:

```
SELECT pname FROM parts
where pname like 'Land%'
```

b) Inspect the execution visually by looking at "QueryStats" and "Executionplan" that you can select in right side or result window.

Put an index on column pname in the parts table and repeat the above steps.
Try with different search criteria and inspect the execution results visually.

Exercise 4 – Prepared Statements

a) Create a new web project in netbeans called JDBCdemo. Create a Customer class and a CustomerMapper class.

b) The Customer class must have the same attributes as the customer table in the mailorder database. The CustomerMapper must have the following methods:

```
Public List<Customer> getAllCustomers();
Public Customer getCustomerByName(String name);
public List<Customer> getCustomersByZip(String zip);
```

c) Create your mapper methods in a way so that you can reuse the Prepared Statement object for every call to the method.

d) Create the necessary indices in the customer table, so that your queries can run as fast as possible.

Enjoy!