# 1 Lab 2 PSM, Trigger, JDBC

## 1.1 Goal:

After this lab, student should know how to

1. Create and use Stored procedure and function
2. How to create and use triggers
3. How to access database through JDBC
4. How to prevent SQL injection.

Students should master the following specifics after the lab:

1. Dynamic SQL
2. Cursor /result set
3. SQLSTATE
4. PK, FK, constraints, grant/revoke
5. Transactions

Reference:

1. **MySQL reference**
http://dev.mysql.com/doc/refman/5.5/en/sql-syntax-compound-statements.html
2. **JDBC API Documentation**
http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/

## 1.2 Lab Report

As in lab 1, you need to show the input and output of your work. In this lab, you also need to design some test cases to proof that your code does work correctly.

# 2 Lab Exercise

## 2.1 Part 1. PSM

1. Setup:

1.1. Create three tables:

```
create table lab2_course(
      id char(10) primary key,
      name char(30) not null,
      credit int not null);

create table lab2_student(
      id char(10) primary key,
      name char(10) not null);

create table lab2_takes(
```

```
id char(10),
course_id char(10),
grade char,
primary key (id, course_id),
foreign key (id) references lab2_student(id),
foreign key (course_id) references lab2_course(id))
```

1.2. Insert data for two students and two courses.

```
insert into lab2_course values ('cs4421', "Database",3);
insert into lab2_course values ('cs4461', "Network",3);
insert into lab2_student values ('S001', "Alice");
insert into lab2_student values ('S002', "Mike");
```

2. Create a procedure named `enroll (student_id, course_id)` to enroll student into a class. If you want to edit an exiting procedure, you need to drop it first using SQL command "`drop yourprocedurename`"

2.1. Change delimiter – please remember if you log out from database connection, you need to set the delimiter again.

```
delimiter //
```

2.2. Create procedure

```
create procedure enroll (id char(10), course_id char(10) )
begin
insert into lab2_takes values (id, course_id, Null);
end //
```

2.3. Call procedure first and then examine whether the data is inserted correctly.

```
call enroll('S001','CS4421') //
select * from lab2_takes//
+------+--------+-------+
| id   | c_id   | grade |
+------+--------+-------+
| 3    | cs4421 | NULL  |
+------+--------+-------+
```

2.4. Grant execute permission on the procedure to grader, so grader can run it.

```
mysql> grant  execute on  procedure enroll to
'cs3425gr'@'%'//
Query OK, 0 rows affected (0.03 sec)
```

To see what have been granted to grader, log in as cs3425gr with password cs3425gr, then run the command "show grants for 'cs3425gr'@'%'

3. Create a function `enrolled(course_id)` to return how many students are currently enrolled in the course

3.1. Create the function

```
create function enrolled(c_id char(10))
    returns int
begin
   declare total int;
    select count(*) into total from lab2_takes where course_id=
c_id;
    return total;
end //
```

3.2. Call function enrolled and check if it returns the correct result

```
select enrolled('cs4421')//
select enrolled('cs4461')//

TODO: select … from …
```

3.3. Give grader the permission to execute the function

```
grant execute on function enrolled to 'cs3425gr'@'%'//
```

## 2.2 Part 2. Trigger

1. Create a trigger to update the total credit when grade is updated

1.1. Add column total_credits.

```
alter table lab2_student add total_credits int default 0;
```

1.2. Create trigger

Please pay attention to how predefined tuple variable OLD and NEW are used.

```
create trigger update_credits
   after update on lab2_takes
   for each row
begin
   if OLD.id = NEW.id and OLD.course_id = NEW.course_id
      and ( OLD.grade is null or OLD.grade = 'F')
      and NEW.grade is not null
      and NEW.grade != 'F' then
              update lab2_student
              set total_credits = total_credits +
                 (select credit
                  from lab2_course
                  where id = OLD.course_id )
              where id=NEW.id;
   end if;
```

```
end//
```

1.3. Update a grade and check if `total_credits` is updated accordingly. You need to design test cases here.

TODO: ... update lab2_takes set grade =....

TODO: mysql> select * from lab2_student where ...

## 2.3 Part3 JDBC

1. Set up lab
   1.1. Create the following table in your database
   lab2_account (account_number char(10) primary key, balance double not null)

   1.2. Insert at least 3 records in the table above.
   Insert into lab2_account values('A001', 100);
   Insert into lab2_account values('B001', 999);
   Insert into lab2_account values('C001', 99.99);

2. Create a new project and setup the JDBC driver in eclipse
   click buildpath -> configure buildpath -> Library -> external library jar ->
   /usr/share/java/mysql -connector-java.jar

3. Connect to database - create new class file JDBCLab.java

```
import java.sql.*;

public class JDBCLab {
	Connection conn = null;
	Statement stmt = null;
	ResultSet rs = null;

	public int connectDB(){
		try {
			conn = DriverManager.getConnection(
				"jdbc:mysql://classdb.it.mtu.edu/YOURDBNAME ",
				"YOURACCOUNT",
				"YOURPASSWORD");
		} catch (SQLException e) {
			System.out.println(e.getMessage());
			e.printStackTrace();
			return 1;
		}
		return 0;
	}

	public void disconnect(){
		try {
			conn.close();
		}
		catch (SQLException ex){
```

```
                    System.out.println("SQLException: " +
                                       ex.getMessage());
                    System.out.println("SQLState: " + ex.getSQLState());
                    System.out.println("VendorError: " +
                                       ex.getErrorCode());
                }

        }



        public static void main(String args[]){
                JDBCLab dblab = new JDBCLab();

                dblab.connectDB();
                dblab.disconnect();
        }
}
```

4. Add select method and call it in main after connecting to db. Run the program.

```
        public void select(){
                Statement stmt = null;
                ResultSet rs = null;

                try {
                        stmt = conn.createStatement();
                        rs = stmt.executeQuery("SELECT account_number,balance
                                                FROM lab2_account");
                        while (rs.next() ) {
                                System.out.println(rs.getString(1)+ ","+
                                                   rs.getString(2));
                        }
                }
                catch (SQLException ex){
                        System.out.println("SQLException: " +
                                           ex.getMessage());
                        System.out.println("SQLState: " + ex.getSQLState());
                        System.out.println("VendorError: " +
                                           ex.getErrorCode());
                }
        }
```

5. Compile and run your program in terminal

   5.1. Open a terminal, then change your directory to your project source dir under your workspace folder

   cd ...

   5.2. Compile your JDBCLab.java

   javac JDBCLab.java

   A file named JDBCLab.class will be generated.

   5.3. Run your program

java -cp ${CLASSPATH}:/usr/share/java/mysql-connector-java.jar JDBCLab

If you don't have $CLASSPATH variable defined, then use this:
java -cp .:/usr/share/java/mysql-connector-java.jar JDBCLab

Please note that you created your own packages, you will need to add your package path to –cp add the package name in front of your JDBCLab. For simplicity, please use default package.

5.4. Modify your code to read username and password from user so you don't have hardcoded password in your source file. Here is the code to read from user. Please notice Console. readPassword() function does not echo the input, so other people will not see what you type.

```
try {
    Console console = System.console();
    if (console == null) {
        System.out.println("console is null. Run the program in terminal");
        System.out.println("For example, first go to bin dir, then run" +
            "java  -cp .:/usr/share/java/mysql-connector-java.jar JDBCLab");
        return;
    }
    username = console.readLine("Please enter your name:");
    password = console.readPassword("Please enter your password:");
}
catch (Exception ex) {
    ex.printStackTrace();
}
```

6. Add withdraw(account_number, amount) method. This method will reduce the balance in account. Use transaction. Call withdraw(… …) in main. Run the program. Please note the following code has a bug in it, but it catches the problem after it checks the return value, and it then rolls back the transaction. **It is your job to figure out the bug and fix it.** In your report, describe what was the bug and how you fixed it.

```
public int withdraw(String account_number, double amount){
        Statement stmt = null;
        ResultSet rs = null;
        int rowcount;

        // start transaction
        try {
            conn.setAutoCommit(false);
            conn.setTransactionIsolation(
                conn.TRANSACTION_SERIALIZABLE);
        } catch (SQLException e) {
            e.printStackTrace();
            return 0;
        }

        try {
            stmt = conn.createStatement();
            rowcount = stmt.executeUpdate(
                "update lab2_account set balance = balance - "
                + amount);
            if (rowcount !=1) {
```

```
                        System.out.println("Something is wrong.
                              You are updateing " + rowcount + "rows");
                        conn.rollback();
                  } else {
                        System.out.println("updated " +
                              rowcount + "rows");
                        conn.commit();
                  }

            }
            catch (SQLException ex){
                  // handle any errors
                  System.out.println("SQLException: " +
                              ex.getMessage());
                  System.out.println("SQLState: " + ex.getSQLState());
                  System.out.println("VendorError: " +
                              ex.getErrorCode());
            }

            return 1;
      }
```

```
      public static void main(String args[]){
            JDBCLab dblab = new JDBCLab();

            dblab.connectDB();
            dblab.select();
            dblab.withdraw('A001',100);
            dblab.disconnect();
      }
```

7. Experience with SQlInjection

   7.1. Suppose a user wanted to make account 'A001' lost money. When he was prompted to type user name, he typed a fake account a' or account_number = 'A001' #. We can simulate this problem by calling with this argument:

```
dblab.withdraw("a' or account_number = 'A001'#",10);
```

   Did the above trick worked? Check if A001 lost 10 dollars.

   7.2. Use dynamic sql to prevent this injection. – That is the whole input from the user should be interpreted as account number only, not something else. Update your code above as below:

```
      PreparedStatement stmt = null;

      ...

      try {
            stmt = conn.prepareStatement(
                  "update lab2_account set balance = balance - "
```

```
                    + amount + "where account_number = ?");
            stmt.setString(1,account_number);
            rowcount = stmt.executeUpdate();
            if (rowcount !=1) {
                  System.out.println(
                        "Something is wrong. You are updateing "
                        + rowcount + "rows");
                  conn.rollback();
            } else {
                  System.out.println("updated " + rowcount +
                                    "rows");
                  conn.commit();
            }

      }
```

7.3. Now suppose a user wanted to make account 'A001' lost money. When he was prompted to type user name, he typed a fake account a' or account_number = 'A001' #. We can simulate this problem by calling with this argument.

```
dblab.withdraw2("a' or account_number = 'A001'#",10);
```

What is the result of running the above? Did the A100 lost 10 dollars?