SOEN 387
WEB-BASED ENTERPRISE APPLICATIONS DESIGN

TUTORIAL – 5

Using Databases

Agenda

- What is Database ?
- What is Database system ?
- Why Use a Database System?
- SQL : an example
- SQL: COMMIT and ROLLBACK
- BLOBS (Binary Large Objects)
- Insert images into a MySQL table with BLOB type
- JDBC (Java Database Connectivity)
    - Configuration
    - Database creation
    - Connectors
    - Libraries
    - Demo Project
- Query execution Best Practices
- SQL output parameters in stored procedure
- Exercise Query Execution using JDBC
- How to prevent SQL injection?

What is Database ?

A large and persistent collection of (more-or-less similar) pieces of information organized in a way that facilitates efficient retrieval and modification

The structure of the database is determined by the abstract data model that is used

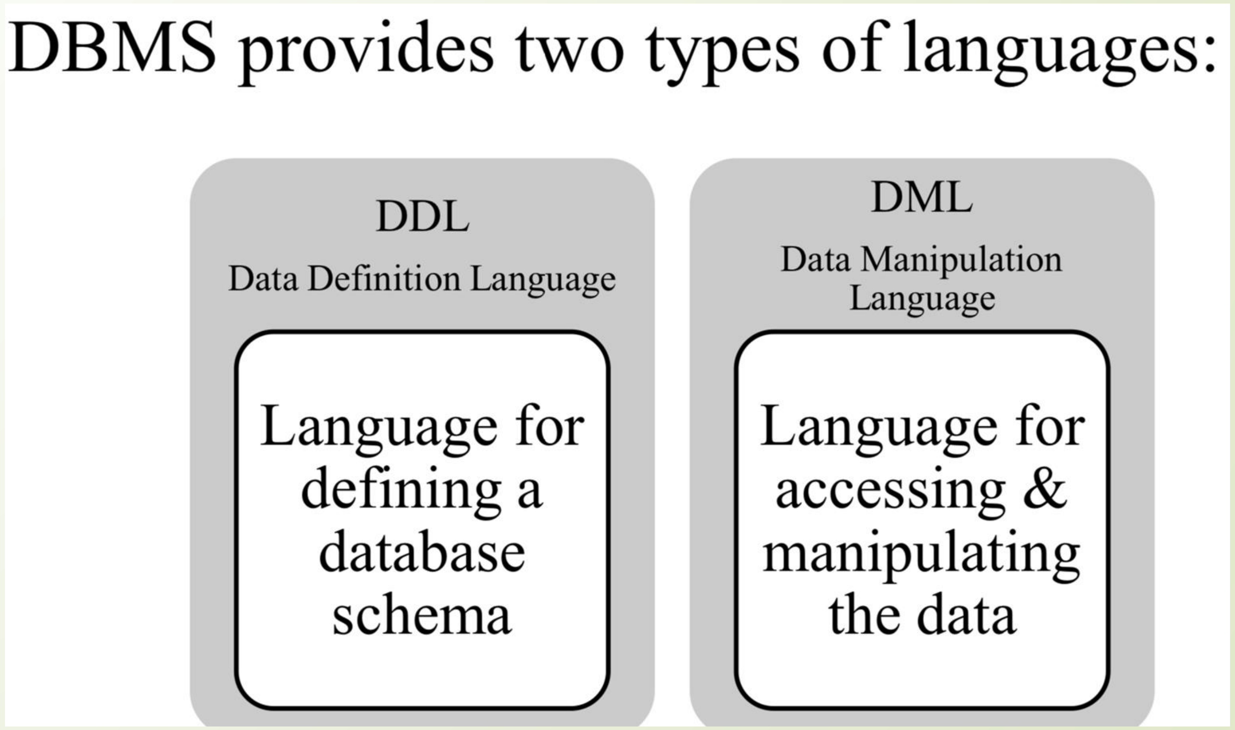
Examples:

1. List of names, addresses, and phone numbers of your friends
2. Information about employees, departments, salaries, managers, etc. in a COMPANY
3. Information about students, courses, grades, professors, etc. in a UNIVERSITY
4. Information about books, users, etc. in a LIBRARY

What is Database system ?

Database Management System (DBMS)

A program (or set of programs) that manages details related to storage and access for a database.


DBMS provides two types of languages:

DDL
Data Definition Language
Language for defining a database schema
DML
Data Manipulation Language
Language for accessing & manipulating the data

Why Use a Database System?

Database systems have concentrated on providing solutions for all of these issues for scaling up Web applications

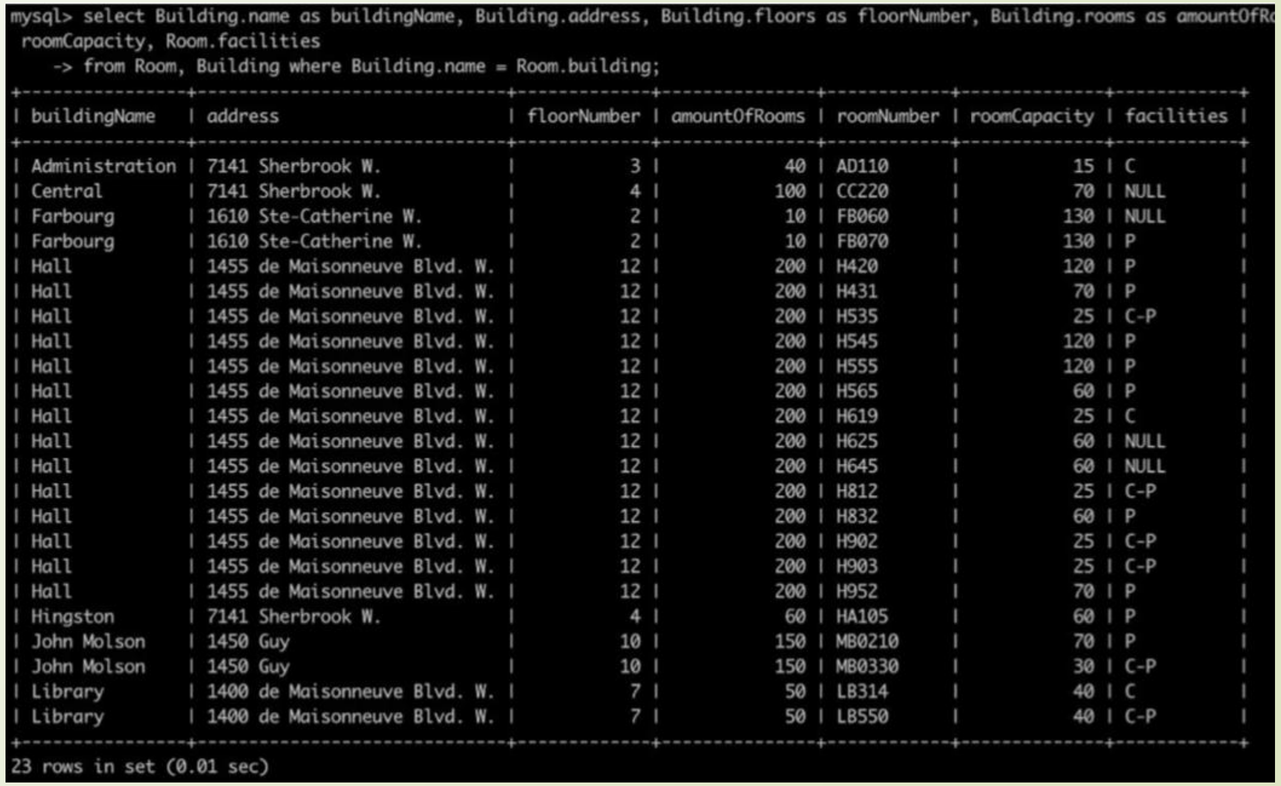
- –Performance
- –Scalability
- –Maintenance
- –Data Integrity
- –Transaction support

While systems differ in their support, most offer some support for all of these.

SQL : An Example

Assume we have database for Concordia in which we have all the information of rooms and address etc.

We wish to know sitting capacity of concordia in various buildings.



```
mysql> select Building.name as buildingName, Building.address, Building.floors as floorNumber, Building.rooms as amountOfRo
 roomCapacity, Room.facilities
    -> from Room, Building where Building.name = Room.building;
+----------------+------------------------------+-------------+---------------+------------+--------------+------------+
| buildingName   | address                      | floorNumber | amountOfRooms | roomNumber | roomCapacity | facilities |
+----------------+------------------------------+-------------+---------------+------------+--------------+------------+
| Administration | 7141 Sherbrook W.            |           3 |            40 | AD110      |           15 | C          |
| Central        | 7141 Sherbrook W.            |           4 |           100 | CC220      |           70 | NULL       |
| Farbourg       | 1610 Ste-Catherine W.        |           2 |            10 | FB060      |          130 | NULL       |
| Farbourg       | 1610 Ste-Catherine W.        |           2 |            10 | FB070      |          130 | P          |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H420       |          120 | P          |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H431       |           70 | P          |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H535       |           25 | C-P        |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H545       |          120 | P          |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H555       |          120 | P          |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H565       |           60 | P          |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H619       |           25 | C          |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H625       |           60 | NULL       |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H645       |           60 | NULL       |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H812       |           25 | C-P        |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H832       |           60 | P          |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H902       |           25 | C-P        |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H903       |           25 | C-P        |
| Hall           | 1455 de Maisonneuve Blvd. W. |          12 |           200 | H952       |           70 | P          |
| Hingston       | 7141 Sherbrook W.            |           4 |            60 | HA105      |           60 | P          |
| John Molson    | 1450 Guy                     |          10 |           150 | MB0210     |           70 | P          |
| John Molson    | 1450 Guy                     |          10 |           150 | MB0330     |           30 | C-P        |
| Library        | 1400 de Maisonneuve Blvd. W. |           7 |            50 | LB314      |           40 | C          |
| Library        | 1400 de Maisonneuve Blvd. W. |           7 |            50 | LB550      |           40 | C-P        |
+----------------+------------------------------+-------------+---------------+------------+--------------+------------+
23 rows in set (0.01 sec)
```

SQL >

```
select Building.name as buildingName, Building.address, Building.floors as floorNumber, Building.rooms as amountOfRooms, Room.roomNumber, Room.capacity as roomCapacity,
Room.facilities from Room, Building where
Building.name = Room.building;
```

SQL: COMMIT and ROLLBACK

The COMMIT command

The transactional command used to save changes invoked by a transaction to the database.

The syntax for the COMMIT command is as follows: COMMIT;

```
SQL> DELETE FROM CUSTOMERS
   WHERE AGE = 25;
SQL> COMMIT;
```

The ROLLBACK Command

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for a ROLLBACK command is as follows – ROLLBACK;

```
SQL> DELETE FROM CUSTOMERS
   WHERE AGE = 25;
SQL> ROLLBACK;
```

BLOBS(Binary Large Objects)

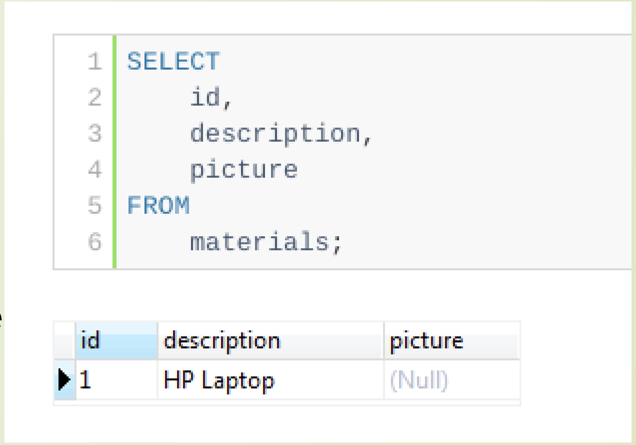
- A BLOB is a binary large object that can hold a variable amount of data.
- It Stores any kind of data in binary format such as images, audio, and video.
- BLOB allocates spaces in Giga Bytes.
- Some projects require a large string or block of binary data to be stored in a database.
- For example, a digital file containing a picture, video, or a song can be stored in a database using a BLOB.

Insert images into a MySQL table with BLOB type

- Suppose you have a table created, then use alter command to add a column of blob type.

- ALTER TABLE materials ADD COLUMN picture blob

- To update the picture column with the data from the picture
  - First, prepare an UPDATE statement.
  - Next, connect to the SQLite database to get the Connection object.
  - Then, create a PreparedStatement object from the Connection object.
  - After that, supply the values to the corresponding parameters using the set* methods of the PreparedStatement object.
  - Finally, execute the UPDATE statement by calling the executeUpdate() method of the PreparedStatement object.



```
SELECT
    id,
    description,
    picture
FROM
    materials;
```

id	description	picture
1	HP Laptop	(Null)

- Refer to this link for complete code and worflow of insertion and query blobs.

Java Database Connectivity

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access any kind of tabular data, especially relational database. It is part of Java Standard Edition platform, from Oracle Corporation. It acts as a middle layer interface between java applications and database.

The JDBC classes are contained in the Java Package **java.sql** and **javax.sql**

JDBC helps you to write Java applications that manage these three programming activities:

- Connect to a data source, like a database.
- Send queries and update statements to the database
- Retrieve and process the results received from the database in answer to your query

JDBC : Configuration

Step1 : Correct tools and configurations.

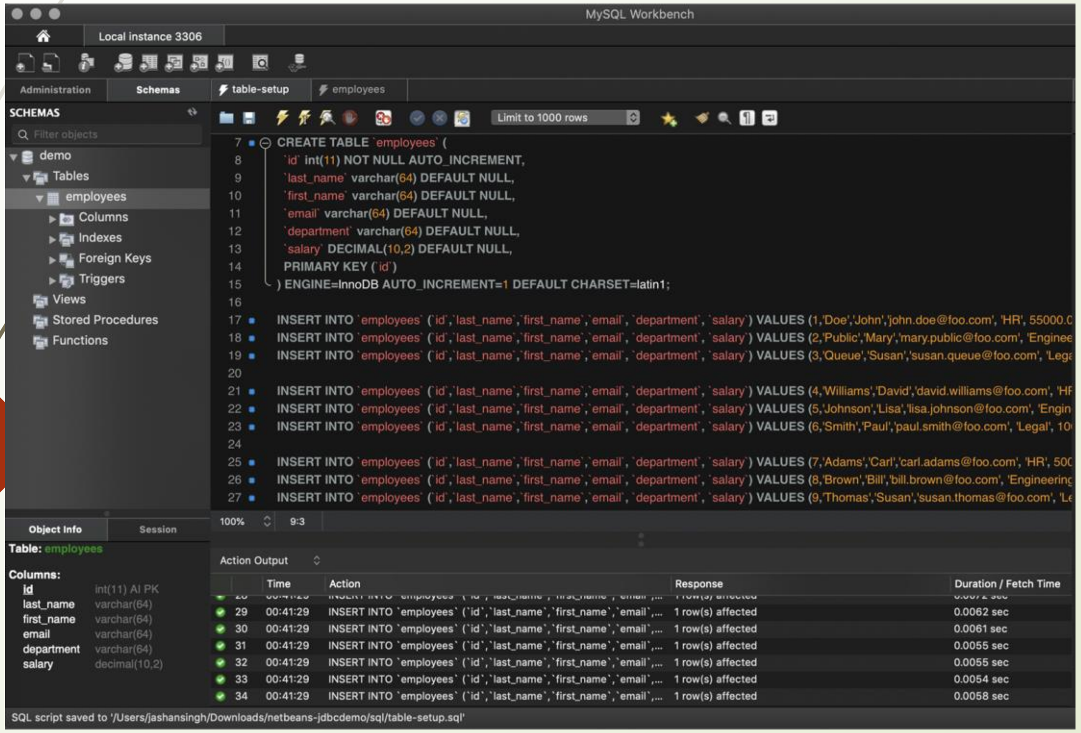
What We Need to configure and create a demo JAVA JDBC project are as following :

- NetBeans 8.2
- JDK 8
- MySQL 5
  - MySQL WorkBench (Not required but recommended)

JDBC : Database creation

Step 2 : Review DataBase Tables

Create a Database Demo with Employee table in it with some information.



```
CREATE TABLE `employees` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `last_name` varchar(64) DEFAULT NULL,
  `first_name` varchar(64) DEFAULT NULL,
  `email` varchar(64) DEFAULT NULL,
  `department` varchar(64) DEFAULT NULL,
  `salary` DECIMAL(10,2) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;

INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`, `department`, `salary`) VALUES (1,'Doe','John','john.doe@foo.com', 'HR', 55000.0
INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`, `department`, `salary`) VALUES (2,'Public','Mary','mary.public@foo.com', 'Enginee
INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`, `department`, `salary`) VALUES (3,'Queue','Susan','susan.queue@foo.com', 'Lega

INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`, `department`, `salary`) VALUES (4,'Williams','David','david.williams@foo.com', 'HF
INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`, `department`, `salary`) VALUES (5,'Johnson','Lisa','lisa.johnson@foo.com', 'Engin
INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`, `department`, `salary`) VALUES (6,'Smith','Paul','paul.smith@foo.com', 'Legal', 10

INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`, `department`, `salary`) VALUES (7,'Adams','Carl','carl.adams@foo.com', 'HR', 50C
INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`, `department`, `salary`) VALUES (8,'Brown','Bill','bill.brown@foo.com', 'Engineering
INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`, `department`, `salary`) VALUES (9,'Thomas','Susan','susan.thomas@foo.com', 'Le
```

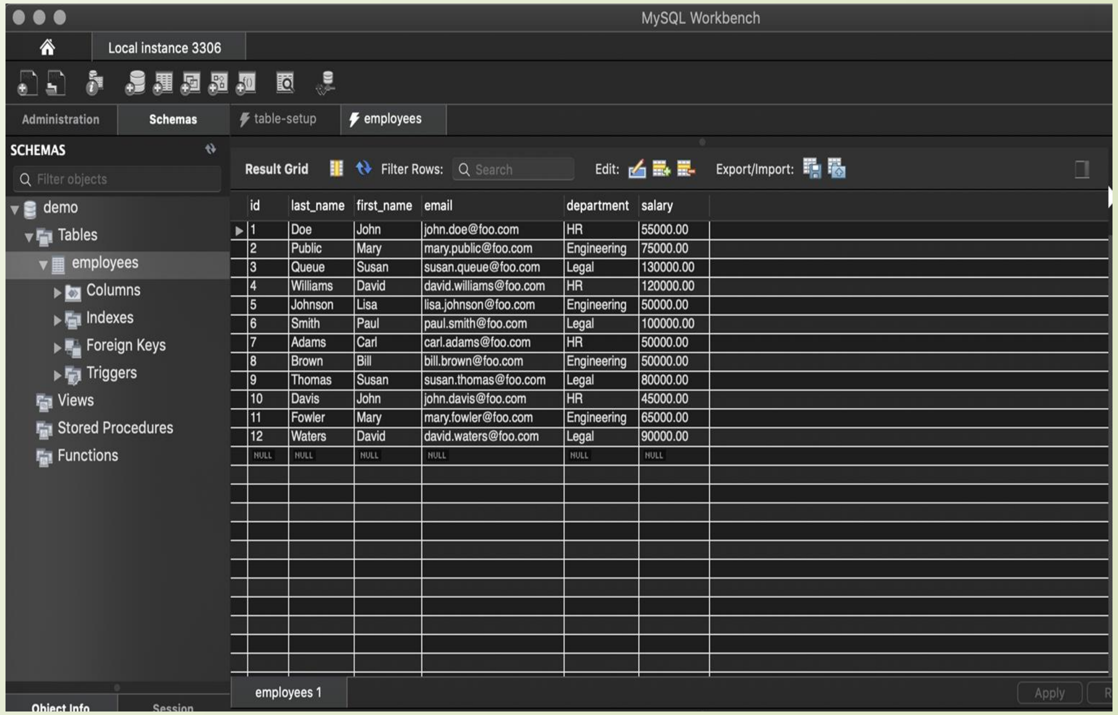
Table: employees

Columns:

id	int(11) AI PK
last_name	varchar(64)
first_name	varchar(64)
email	varchar(64)
department	varchar(64)
salary	decimal(10,2)

```
    Time      Action                                                          Response           Duration / Fetch Time
29  00:41:29  INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`,...  1 row(s) affected  0.0062 sec
30  00:41:29  INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`,...  1 row(s) affected  0.0061 sec
31  00:41:29  INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`,...  1 row(s) affected  0.0055 sec
32  00:41:29  INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`,...  1 row(s) affected  0.0055 sec
33  00:41:29  INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`,...  1 row(s) affected  0.0054 sec
34  00:41:29  INSERT INTO `employees` (`id`,`last_name`,`first_name`,`email`,...  1 row(s) affected  0.0058 sec
```

SQL script saved to '/Users/jashansingh/Downloads/netbeans-jdbcdemo/sql/table-setup.sql'



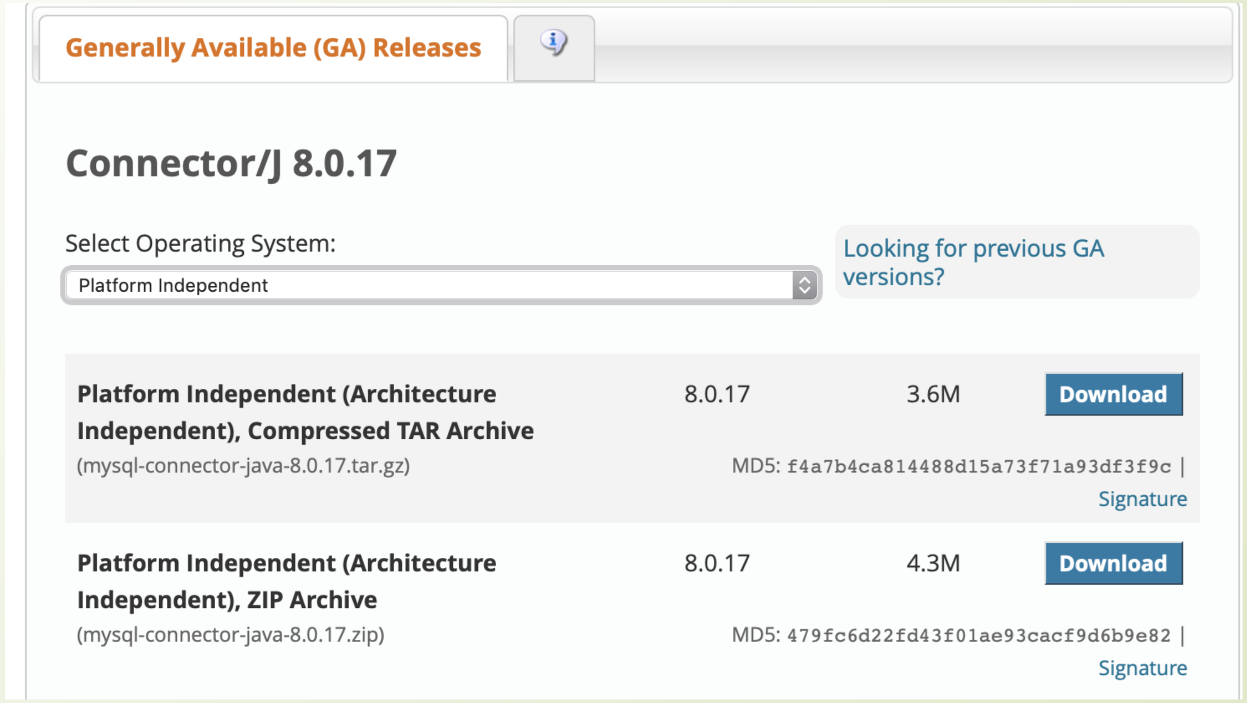
id	last_name	first_name	email	department	salary
1	Doe	John	john.doe@foo.com	HR	55000.00
2	Public	Mary	mary.public@foo.com	Engineering	75000.00
3	Queue	Susan	susan.queue@foo.com	Legal	130000.00
4	Williams	David	david.williams@foo.com	HR	120000.00
5	Johnson	Lisa	lisa.johnson@foo.com	Engineering	50000.00
6	Smith	Paul	paul.smith@foo.com	Legal	100000.00
7	Adams	Carl	carl.adams@foo.com	HR	50000.00
8	Brown	Bill	bill.brown@foo.com	Engineering	50000.00
9	Thomas	Susan	susan.thomas@foo.com	Legal	80000.00
10	Davis	John	john.davis@foo.com	HR	45000.00
11	Fowler	Mary	mary.fowler@foo.com	Engineering	65000.00
12	Waters	David	david.waters@foo.com	Legal	90000.00
NULL	NULL	NULL	NULL	NULL	NULL

JDBC : Connectors

Step 3 : Download the JDBC connectors

Link : https://dev.mysql.com/downloads/connector/j/

In our case we download the platform independent version.


Generally Available (GA) Releases

Connector/J 8.0.17

Select Operating System: Platform Independent

Looking for previous GA versions?

Platform Independent (Architecture Independent), Compressed TAR Archive (mysql-connector-java-8.0.17.tar.gz)	8.0.17	3.6M	Download
MD5: `f4a7b4ca814488d15a73f71a93df3f9c` | Signature			
Platform Independent (Architecture Independent), ZIP Archive (mysql-connector-java-8.0.17.zip)	8.0.17	4.3M	Download
MD5: `479fc6d22fd43f01ae93cacf9d6b9e82` | Signature			

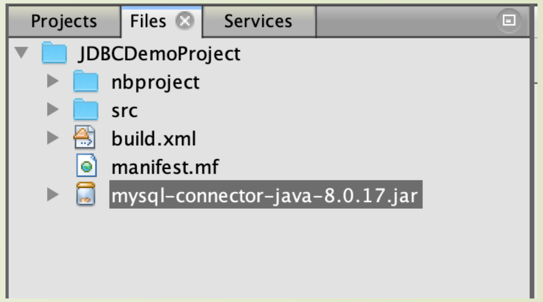
JDBC : Libraries

Step 4 : Create a Demo Java Project in NetBeans

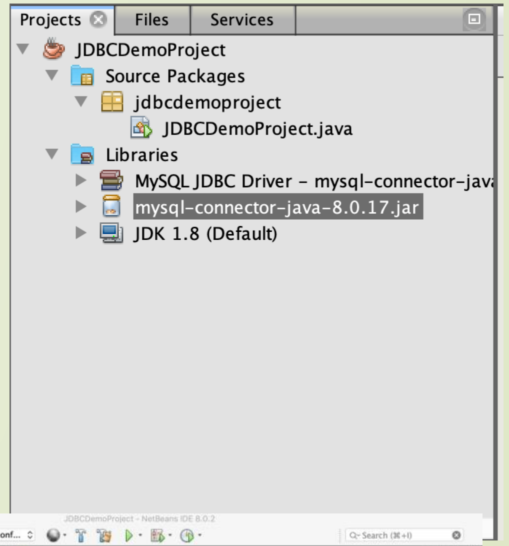
1. Add the mysql-connector-java-8.0.17.jar to the project files.
2. Add Jar into the project Libraries.

OR

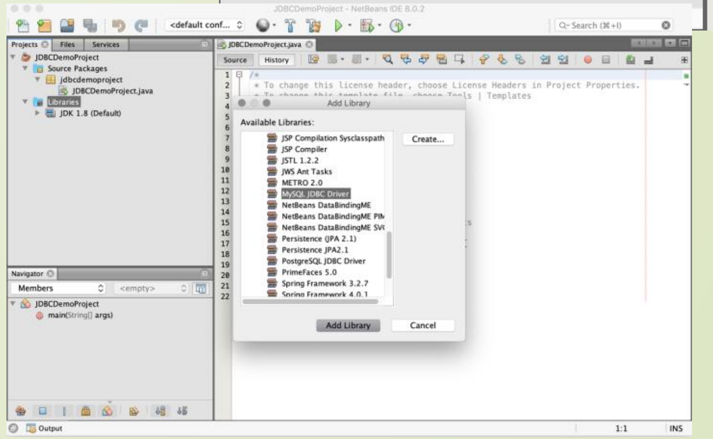
Step 4. After creating a Demo Java Project, right click on libraries > Add Library > mysql connector > Click add.



```
JDBCDemoProject
  nbproject
  src
  build.xml
  manifest.mf
  mysql-connector-java-8.0.17.jar
```



```
JDBCDemoProject
  Source Packages
    jdbcdemoproject
      JDBCDemoProject.java
  Libraries
    MySQL JDBC Driver - mysql-connector-java
    mysql-connector-java-8.0.17.jar
    JDK 1.8 (Default)
```



```
JDBCDemoProject
  Source Packages
    jdbcdemoproject
      JDBCDemoProject.java
  Libraries
    JDK 1.8 (Default)
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
```

Add Library

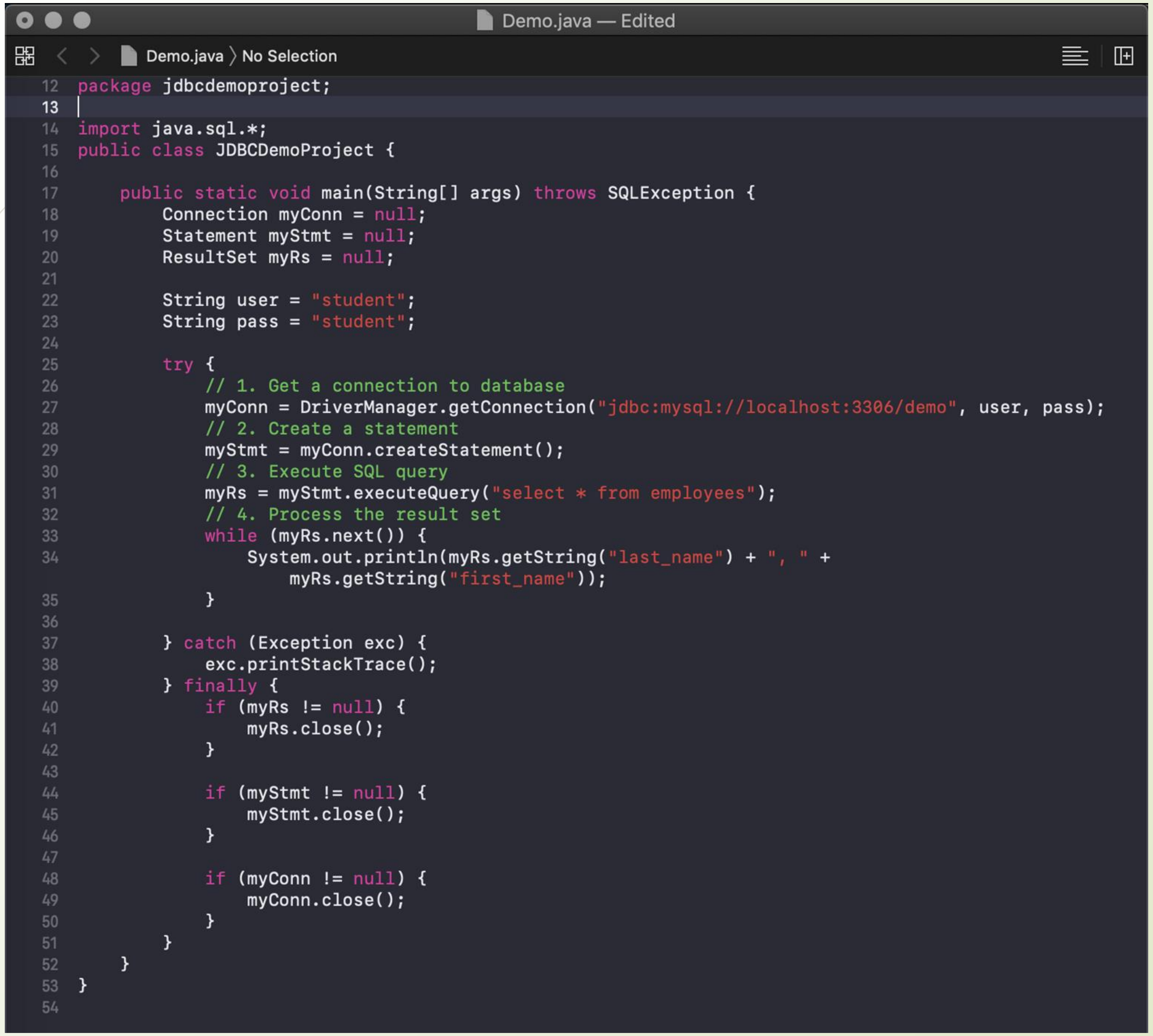
Available Libraries:

```
JSP Compilation Sysclasspath
JSP Compiler
JSTL 1.2.2
JWS Ant Tasks
METRO 2.0
MySQL JDBC Driver
NetBeans DataBindingME
NetBeans DataBindingME PIM
NetBeans DataBindingME SVG
Persistence (JPA 2.1)
Persistence JPA2.1
PostgreSQL JDBC Driver
PrimeFaces 5.0
Spring Framework 3.2.7
Spring Framework 4.0.1
```

Create... | Add Library | Cancel

```
JDBCDemoProject
  main(String[] args)
```

JDBC : Demo Project



```
package jdbcdemoproject;

import java.sql.*;
public class JDBCDemoProject {

    public static void main(String[] args) throws SQLException {
        Connection myConn = null;
        Statement myStmt = null;
        ResultSet myRs = null;

        String user = "student";
        String pass = "student";

        try {
            // 1. Get a connection to database
            myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/demo", user, pass);
            // 2. Create a statement
            myStmt = myConn.createStatement();
            // 3. Execute SQL query
            myRs = myStmt.executeQuery("select * from employees");
            // 4. Process the result set
            while (myRs.next()) {
                System.out.println(myRs.getString("last_name") + ", " +
                    myRs.getString("first_name"));
            }

        } catch (Exception exc) {
            exc.printStackTrace();
        } finally {
            if (myRs != null) {
                myRs.close();
            }

            if (myStmt != null) {
                myStmt.close();
            }

            if (myConn != null) {
                myConn.close();
            }
        }
    }
}
```

Query Execution Best Practices

- Avoid hardcoding server or host address. (Hint: See Configuration)
- Try with resource statement: The try-with-resources statement is a try statement that declares one or more resources. A resource is an object that must be closed after the program is finished with it. (See here)
- Avoid using SELECT * always because:
  - you don't need all the columns
  - Columns can change
  - Columns can be added/removed.
- Protect JDBC application against SQL Injection (See article)

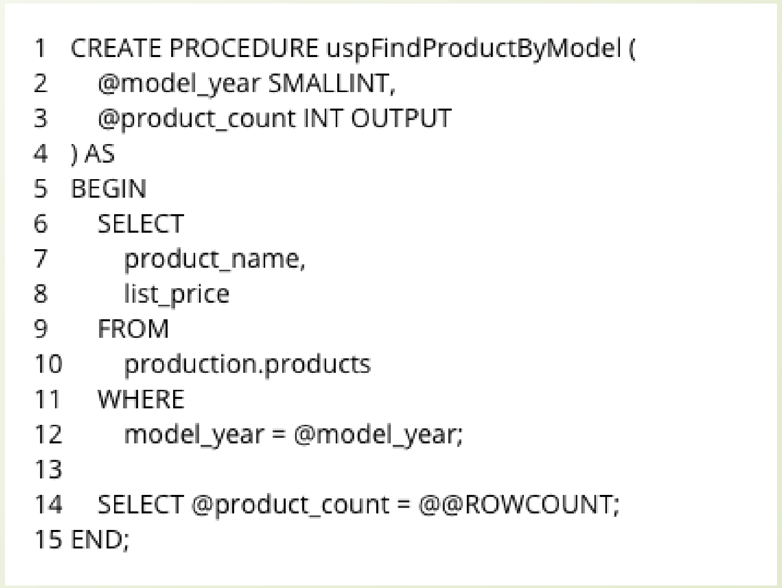
SQL output parameters in stored procedure

The Output Parameters in Stored Procedures are used to return some value or values. A Stored Procedure can have any number of output parameters.

You should know how to use the output parameters to return data back to the calling program.

To create an output parameter for a stored procedure, you use the following syntax: parameter_name data_type OUTPUT

For example, the following stored procedure finds products by model year and returns the number of products via the @product_count output parameter:



```
CREATE PROCEDURE uspFindProductByModel (
    @model_year SMALLINT,
    @product_count INT OUTPUT
) AS
BEGIN
    SELECT
        product_name,
        list_price
    FROM
        production.products
    WHERE
        model_year = @model_year;

    SELECT @product_count = @@ROWCOUNT;
END;
```

Check this cool video on SQL output parameter with SP

Here, create an output parameter named @product_count to store the number of products found:

@product_count INT OUTPUT

Second, after the SELECT statement, assign the number of rows returned by the query(@@ROWCOUNT) to the @product_count parameter.

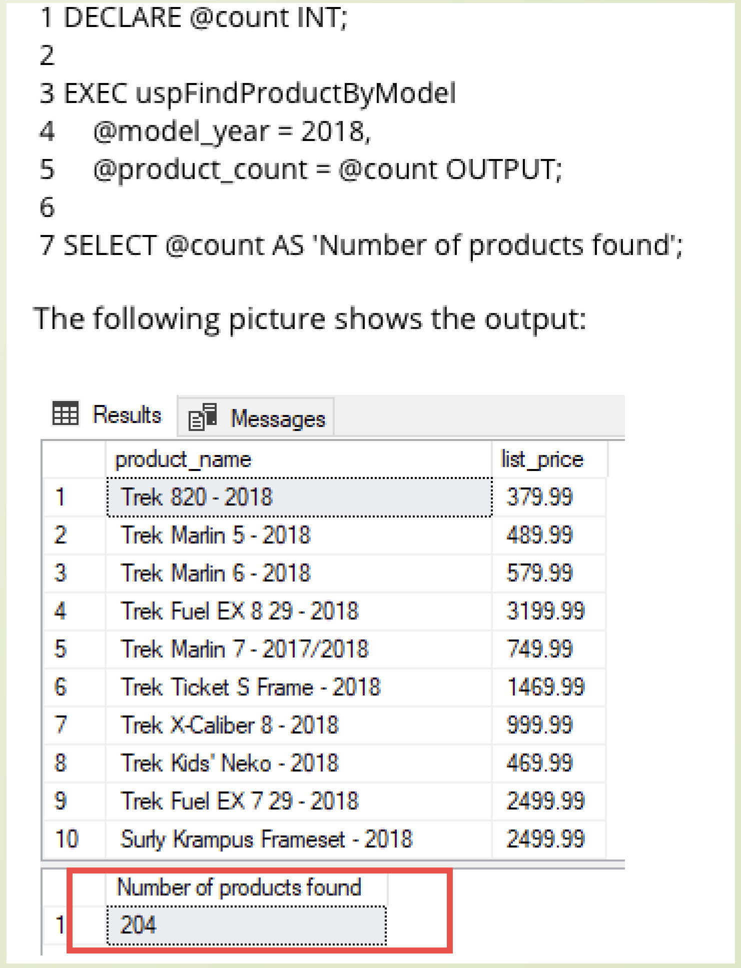
Once the CREATE PROCEDURE statement is executed, the uspFindProductByModel stored procedure is compiled and saved in the database catalog.

Calling stored procedures with output parameters:

First, declare variables to hold the value returned by the output parameters.

Second, use these variables in the stored procedure call.

For example, the following statement executes the uspFindProductByModel stored procedure:



```sql
DECLARE @count INT;

EXEC uspFindProductByModel
    @model_year = 2018,
    @product_count = @count OUTPUT;

SELECT @count AS 'Number of products found';
```

The following picture shows the output:

Results | Messages

	product_name	list_price
1	Trek 820 - 2018	379.99
2	Trek Marlin 5 - 2018	489.99
3	Trek Marlin 6 - 2018	579.99
4	Trek Fuel EX 8 29 - 2018	3199.99
5	Trek Marlin 7 - 2017/2018	749.99
6	Trek Ticket S Frame - 2018	1469.99
7	Trek X-Caliber 8 - 2018	999.99
8	Trek Kids' Neko - 2018	469.99
9	Trek Fuel EX 7 29 - 2018	2499.99
10	Surly Krampus Frameset - 2018	2499.99

	Number of products found
1	204

Exercise: Query Execution using JDBC

- Create a database connection using JDBC by passing server URL, username and password.

- Use a scanner to take user inputs for student name, roll no and class.

- Execute INSERT INTO statement. Hint(`String sql = "insert into student1 values('"+name+"',"+roll+",'"+cls+"')";`)
    - Make use of try catch clock to handle any exception
    - Try block should contain reference to connection interface. (Hint: `con = DriverManager.getConnection(url,user,pass);`)
    - Fire the SQL query. Hint: (`Statement st = con.createStatement(); int m = st.executeUpdate(sql);`

- Execute SELECT FROM statement to see the tuples inserted.

- Execute DELETE FROM statement to deleted a tuple.

How to prevent SQL injection?

Primary Defenses:

- Use of Prepared Statements (with Parameterized Queries)
- Use of Stored Procedures
- Escaping All User Supplied Input

The following code example uses a PreparedStatement, Java's implementation of a parameterized query, to execute the same database query.

```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

Further reading : Link1

Link2