JDBC Basics

In this video:

- Basic concepts of JDBC
- Important Classes and Interfaces used
- CRUD Operations

What is JDBC?

Full Form - Java Database Connectivity

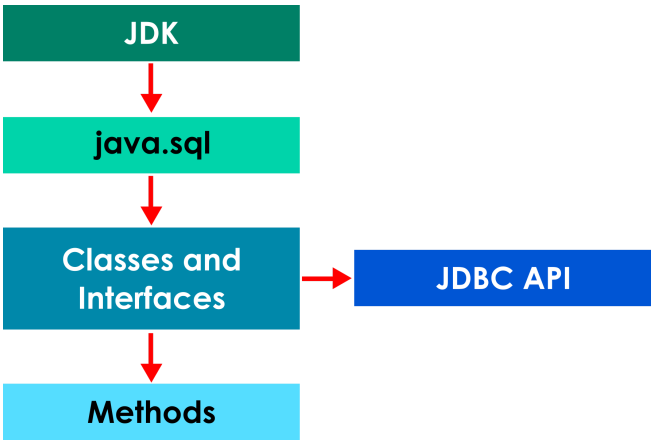
- It is a Java API used for connecting your Java application with a database.
- Used to access tabular data in any relational database.

API - Application Programming Interface

- It is a set of classes and interfaces used to achieve a particular feature.
- Here, the feature is nothing but database connectivity.

JDBC API -> Part of JDK Installation -> Available as the java.sql package

- Use or import the java.sql package whenever writing JDBC Programs.



Database Connectivity:

- Java Application has the JDBC API.
- But there are many database products with different architectures (MySQL, Oracle, MS-SQL etc.)
- The database product takes responsibility in providing something called a JDBC Driver according to its architecture.
- The driver acts like an interface between the application and the database.

- The JDBC Driver is nothing but a .jar file.
- It can be installed from the specific database product website or installer.
- It must be linked to the Java application in order for database connectivity to happen.

Pre-requisites:

For Java:

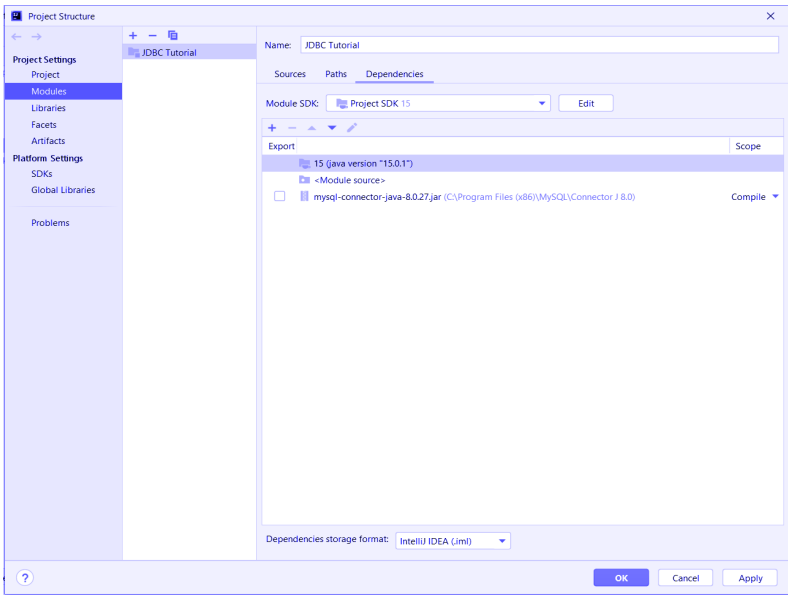
- Java JDK - I use 15, but you can use the latest version
- An IDE - IntelliJ IDEA (or Eclipse, Netbeans etc.)

For Database:

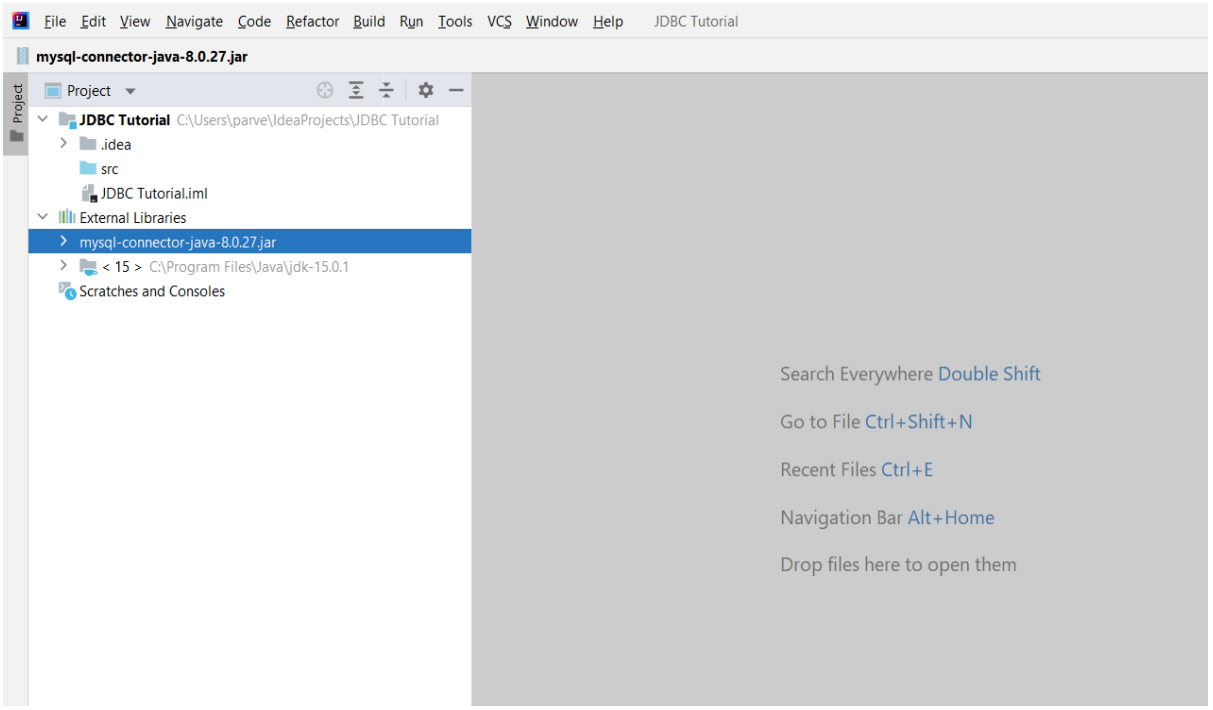
- MySQL Server 8 - free, open source, easy to learn and implement
- A client software - MySQL Shell (or cmd, Workbench etc.)
- MySQL Connector (JDBC Driver)

Linking the Driver:

- Open MySQL Installer and download the JDBC Driver.
- Open IntelliJ and create a new Java project.
- To link the driver,
  Go to File -> Project Structure -> Dependencies -> Add -> Jars -> Select the driver (jar file) -> Apply and OK



- Go to External libraries and you can see that our driver has been connected.



Database Connectivity Steps:

- ➔ Import java.sql package
- ➔ Connect your application to the database
  - ◆ DriverManager Class
  - ◆ Connection Interface
- ➔ Execute queries
  - ◆ Statement Interface
- ➔ Store and retrieve data
  - ◆ ResultSet Interface
- ➔ Close the connection

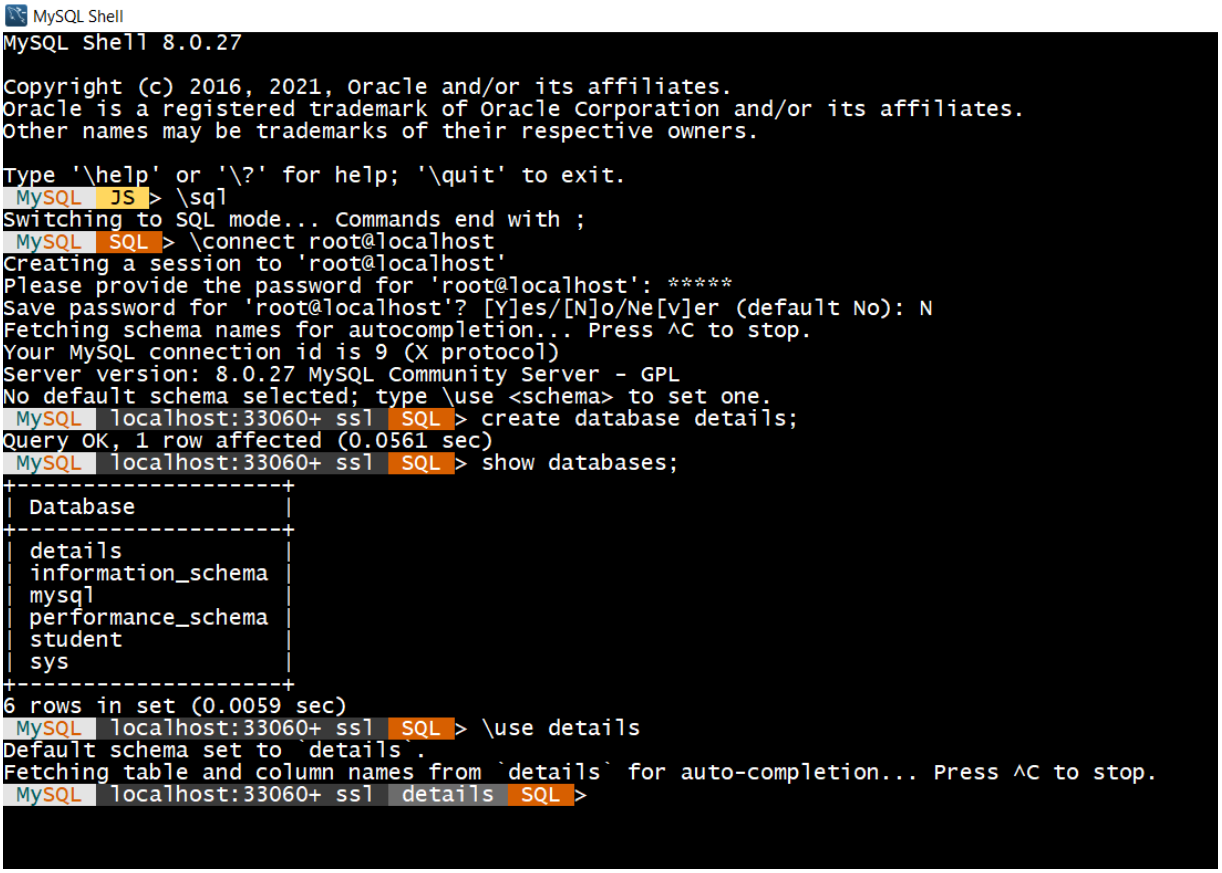
A result set is nothing but the tabular data present inside the table.

Also, always use a try-catch block when writing JDBC programs because the connection between your application and the database happens during the runtime. So to avoid exceptions we always use the try-catch block.

Database Design:

- Open MySQL Shell.
- Change to SQL mode using \sql
- Connect to the server by providing \connect root@localhost
- Enter the root password and type 'N' for save password question

- To view the list of databases, type show databases; command
- To create a new database, type create database database_name;
- To use a database type \use database_name



- Here, I have created a database named details.
- Within details let us create a table.

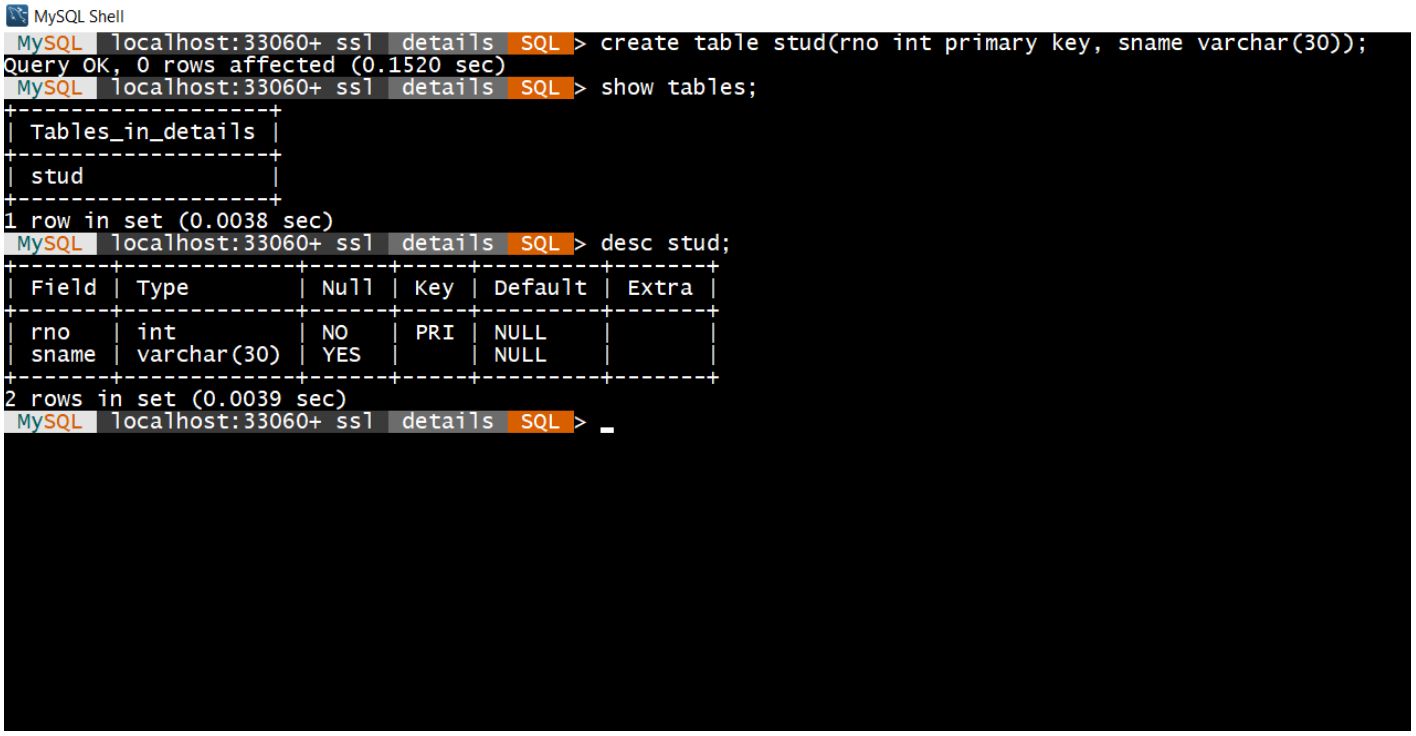
Table Creation:

- To create a table, use the create table command.

```
create table stud(rno int primary key, sname varchar(30));
```

- Here, we have created a table named stud with 2 columns, rno of integer type and sname of varchar type.
- The rno column is a primary key which allows only unique values and doesn't allow null values.
- Use the show tables; command to display the list of tables in the database.
- Type desc table_name; to describe the table name.

The data to be entered can be done using MySQL Shell. But we would do that using JDBC.



Creating a connection:

- To create a connection we use the DriverManager class and we store the connection within the Connection interface.
- The getConnection( ) method from the DriverManager class is used to create and return a connection.
- We provide three arguments inside this method.
    - Connection URL - Specific to different databases
    - Username - root -> default username
    - Password - your username's password
- We store the returned connection inside a Connection object.

```
Connection c = DriverManager.getConnection(cs, user, pass)
```

Connection URL for MySQL -> jdbc:mysql://localhost:3306/database-name

jdbc -> API
mysql -> Database product
localhost -> server name
3306 -> port number

And we provide the database name to be connected as well.

Other Connection URLs:

Oracle -> jdbc:oracle:thin:@//myoracle.db.server:1521/my_servicename
PostgreSQL -> jdbc:postgresql://host:port/database?properties

Example Program:

```
import java.sql.*;

public class DatabaseConnect
{
public static void main(String[] args)
{
try
{
    String url = "jdbc:mysql://localhost:3306/details";
    String user = "root";
    String pass = "mysql";

    Connection con = DriverManager.getConnection(url,user,pass);
    System.out.println("Database connected successfully");
    con.close();
}
catch (Exception ex)
{
    System.out.println("Not connected");
}
}
}
```

Executing Queries:

- After creating a connection we can start executing queries using our Java application.
- To allow execution of queries we use the createStatement( ) method of the Connection interface. It returns a Statement object.
- We store the returned Statement object inside the Statement interface object.

```
Statement st = c.createStatement( )
```

Methods in Statement:

executeUpdate( ) - Queries which do not return a result set.
Commands like insert, update and delete use this method

executeQuery( ) - Queries which return a result set.
Commands like select, show are examples

Example:

```
String sql = "insert into table values(100, 'User')"
st.executeUpdate(sql)
```

- **The executeUpdate( ) method returns the number of rows affected which can be stored and used if needed.**

Inserting Data:

- **Inserting data in SQL can be done using the insert command.**

```
insert into table_name values(100,'Jack');
insert into table_name values(100,'Jill'),(102,'John');
```

- **The second command is called bulk insert which inserts 2 rows at a time. You can insert as many rows as you want.**

Example Program for Inserting Data:

```
import java.sql.*;

public class InsertData
{
public static void main(String[] args)
{
try
{

     String url = "jdbc:mysql://localhost:3306/details";
     String user = "root";
     String pass = "mysql";

     Connection con = DriverManager.getConnection(url,user,pass);
     Statement stmt = con.createStatement();

     String sql = "insert into stud values (100, 'Jack')";
     stmt.executeUpdate(sql);

     System.out.println("Data inserted successfully");

     con.close();
}
catch (Exception ex)
{

     System.out.println("Data not inserted");
     ex.printStackTrace();
}
}
}
```

The Result Set:

- This stores the tabular data returned by executing a query.
- It also maintains a cursor which points a row in the table.
- We can't directly print the result set. Instead, we first access each row and then we access each data from the columns.

```
String sql = "select * from table"
ResultSet rs = st.executeQuery(sql)
```

Common Methods:

```
next( ) -> For moving through rows
getString( ) -> Access column data of string type
getInt( ) -> Access column data of int type
```

- The cursor by default is pointed above the first row. When the next( ) method is encountered, it checks for a row below. If the row is present the method returns true and the cursor starts pointing to the next row.
- When we use the next( ) method within a while loop, it keeps on updating the cursor to point to all the rows till the last row.
- After the last row, there would be no rows. So, the next( ) method returns false and we exit the loop.

```
while(rs.next( ))
{
        rs.getInt(column_name)
        rs.getString(column_name)
}
```

- We can also provide column number instead of column name within the get( ) methods.
- The column number starts from 1.

100	Jack
101	Jill
102	Mark
103	Kate
104	John

Retrieving Data:

- We use the select command in SQL to retrieve data.

```
select * from table_name;
select column from table_name;
select * from table_name where condition;
```

- We can retrieve the complete table, or particular columns, or particular rows based on a condition.

Example Program:

```
import java.sql.*;

public class RetrieveData
{
public static void main(String[] args)
{
try
{
     String url = "jdbc:mysql://localhost:3306/details";
     String user = "root";
     String pass = "mysql";

     Connection con=DriverManager.getConnection(url,user,pass);
     Statement stmt=con.createStatement();

     String sql = "select * from stud";
     ResultSet rs = stmt.executeQuery(sql);

     while(rs.next())
     {
          int roll = rs.getInt("rno");
          String name = rs.getString("sname");

          System.out.println(roll + "\t" + name);
     }

     con.close();
  }
  catch(Exception e)
  {
      System.out.println("Some error occurred!");
  }
  }
  }
```

Updating Data:

- Data updation can be done using the update command in SQL.

```
update stud set sname = 'Jane' where rno = 100;
```

Example Program:

```
import java.sql.*;

public class UpdateData
{
public static void main(String[] args)
{
try
{
      String url = "jdbc:mysql://localhost:3306/details";
      String user = "root";
      String pass = "mysql";

      Connection con = DriverManager.getConnection(url,user,pass);
      Statement stmt = con.createStatement();

      String sql = "update stud set sname = 'Jane' where rno = 100";
      stmt.executeUpdate(sql);

      System.out.println("Data Updated");
      con.close();
}
catch(Exception e)
{
      System.out.println("Some error occurred!");
}
}
}
```

Deleting Data:

- To delete data, we use the delete command in SQL.

```
delete from stud where rno = 102;
```

Example Program:

```
import java.sql.*;

public class DeleteData
{
```

```
public static void main(String[] args)
{
try
{
      String url = "jdbc:mysql://localhost:3306/details";
      String user = "root";
      String pass = "mysql";

      Connection con=DriverManager.getConnection(url,user,pass);
      Statement stmt=con.createStatement();

      String sql = "delete from stud where rno = 102";
      stmt.executeUpdate(sql);
      System.out.println("Data deleted successfully");
      con.close();
}
catch(Exception e)
{
      System.out.println("Some error occurred!");
}
}
}
```

Summary:

- We learned the basic concepts of JDBC
- Got an introduction to the important classes and interfaces used in JDBC
- Demonstrated the CRUD operations using Java's JDBC API

****