JAVA DATABASE CONNECTIVITY

Prerequisite:

- Core java
- DBMS (SQL,PL SQL, NO SQL)

SQL BASICS

- Structured Query Language (SQL) is a standardized language that allows you to perform operations on a database, such as creating entries, reading content, updating content, and deleting entries.
- **Create Database**

SQL> CREATE DATABASE DATABASE_NAME;

EXAMPLE: SQL> CREATE DATABASE EMP;

- **Drop Database**
- **SQL> DROP DATABASE DATABASE_NAME;**
- **Create Table**

```
SQL> CREATE TABLE table_name
(
  column_name column_data_type,
  column_name column_data_type,
  column_name column_data_type
  ...
);
```

SQL BASICS

- Structured Query Language (SQL) is a standardized language that allows you to perform operations on a database, such as creating entries, reading content, updating content, and deleting entries.
- **Create Database**

```
SQL> CREATE DATABASE DATABASE_NAME;
EXAMPLE: SQL> CREATE DATABASE EMP;
```

- **Drop Database**
- **SQL> DROP DATABASE DATABASE_NAME;**
- **Create Table**

```
SQL> CREATE TABLE table_name
(
  column_name column_data_type,
  column_name column_data_type,
  column_name column_data_type
  ...
);
```

DATABASE: BU_CSE
TABLE NAME: BU_JAVA_MARKS

Name	Roll number	Marks

SQL BASICS

- **Drop Table**

```
SQL> DROP TABLE table_name;
```

- **INSERT Data**

```
SQL> INSERT INTO table_name VALUES (column1, column2, ...);
```

- **SELECT Data**

```
SQL> SELECT column_name, column_name, ...
     FROM table_name
     WHERE conditions;
```

- **UPDATE Data**

```
SQL> UPDATE table_name
     SET column_name = value, column_name = value, ...
    WHERE conditions;
```

- **DELETE Data**
- SQL> DELETE FROM table_name WHERE conditions;

SQL BASICS

- **Drop Table**

SQL> DROP TABLE table_name;

- **INSERT Data**

SQL> INSERT INTO table_name VALUES (column1, column2, ...);

- **SELECT Data**

```
SQL> SELECT column_name, column_name, ...
    FROM table_name
    WHERE conditions;
```

DATABASE: BU_CSE

TABLE NAME: BU_JAVA_MARKS

Name	Roll number	Marks
somesh	123	9
mikesh	456	10

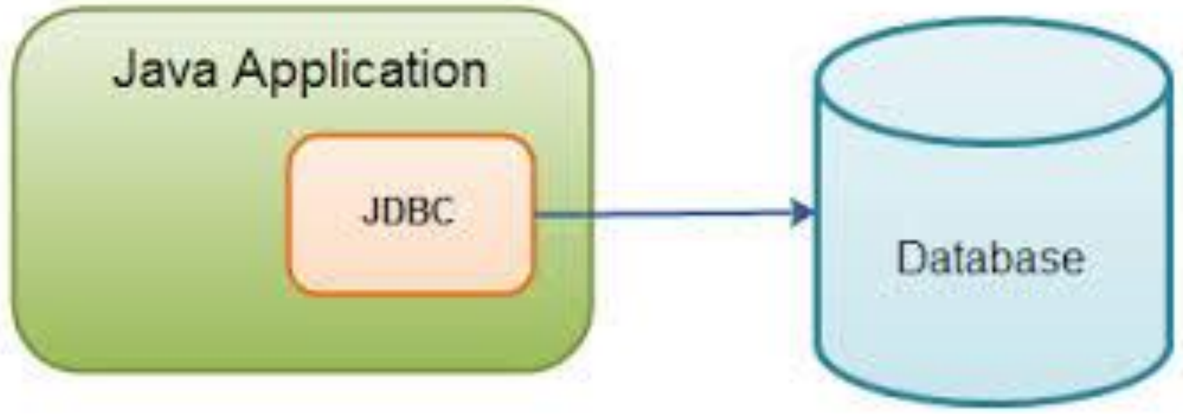
- **UPDATE Data**

```
SQL> UPDATE table_name
    SET column_name = value, column_name = value, ...
   WHERE conditions;
```

- **DELETE Data**
- **SQL> DELETE FROM table_name WHERE conditions;**

Definition:

- JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.
- JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:
- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver



Why to Learn JDBC?

- JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- FOR JAVA APPLICATION DATABASE, SOME CONNECTIVITY IS REQUIRED IS KNOWN AS JDBC
- JDBC GUIDELINES DEFINED BY JAVA VENDOR
- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

1. Making a connection to a database.
2. Creating SQL or MySQL statements.
3. Executing SQL or MySQL queries in the database.
4. Viewing & Modifying the resulting records.

Applications of JDBC

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as −

1. Java Applications
2. Java Applets
3. Java Servlets
4. Java ServerPages (JSPs)
5. Enterprise JavaBeans (EJBs).

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data

Features of JDBC

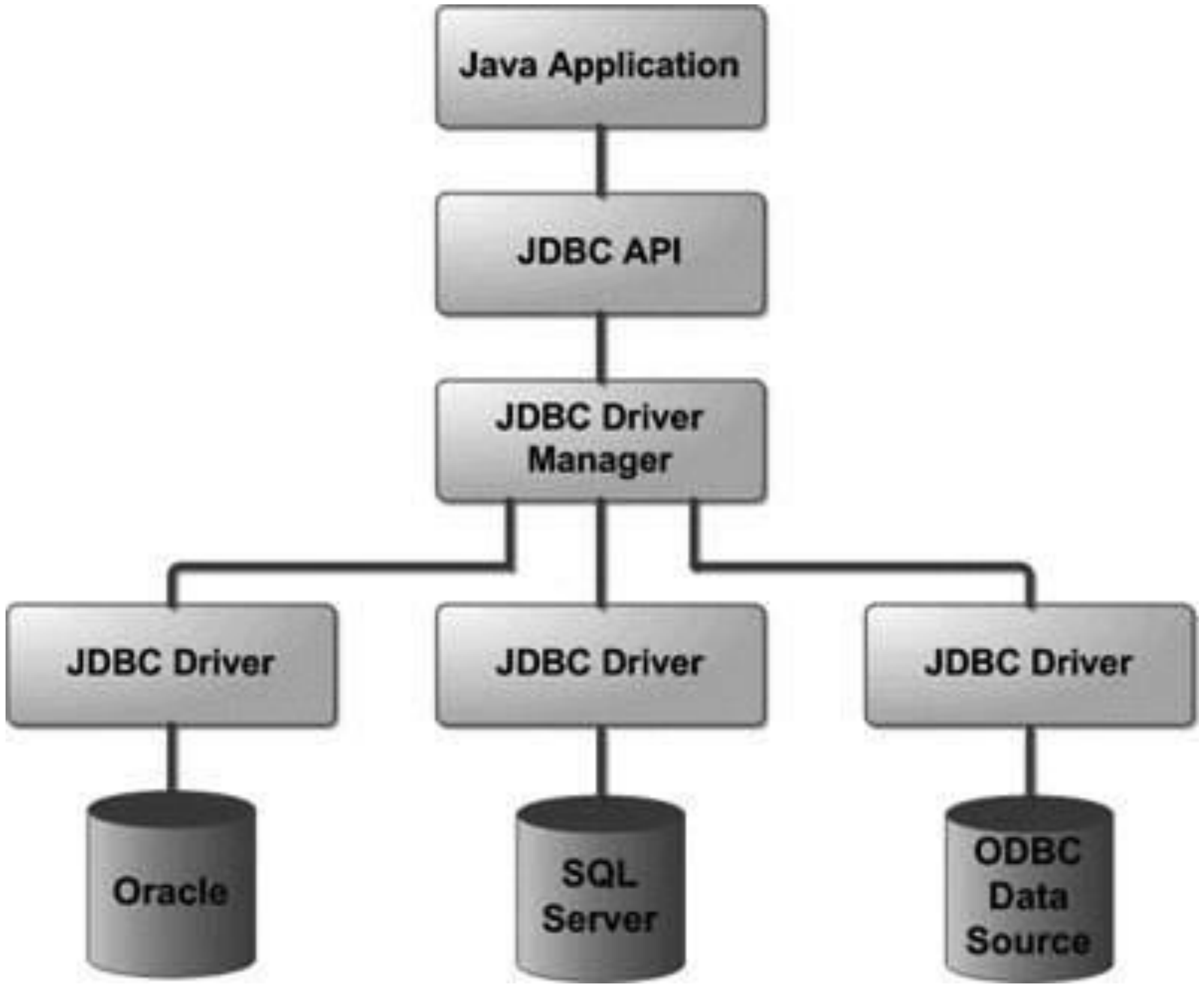
- Database independent API
- Platform independent technology
- We can perform CRUD operation very easily

1. Create 2. Retrieve 3. Update 4. Delete

- **We can also perform complex operation like joins, stored procedure etc.**
- Huge vendor support for JDBC.

https://www.oracle.com/technetwork/java/index-136695.html

JDBC Architecture



Common JDBC Components

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

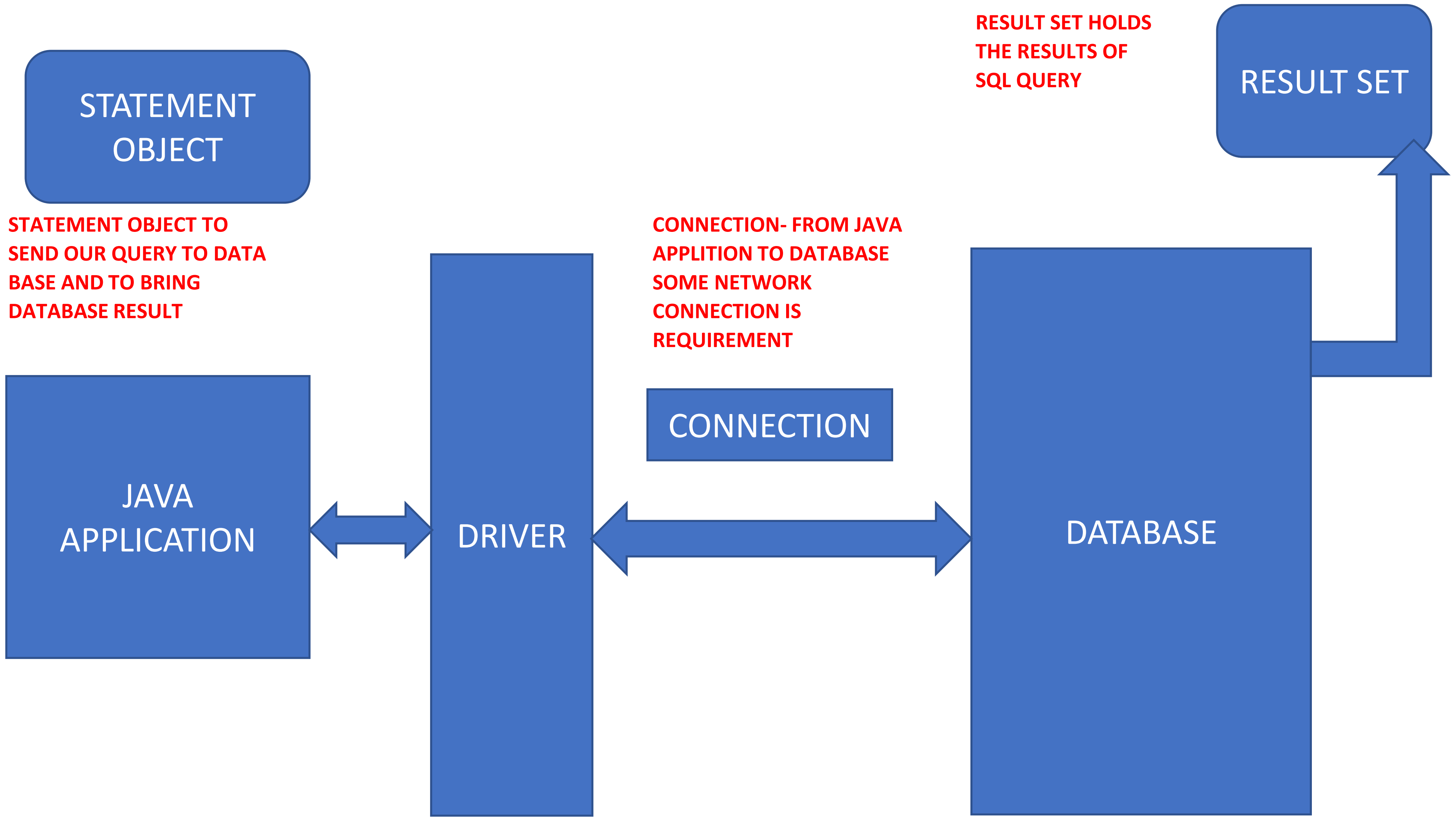
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.

- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

- **Statement**: You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

- **ResultSet**: These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

- **SQLException**: This class handles any errors that occur in a database application.



RESULT SET HOLDS THE RESULTS OF SQL QUERY

STATEMENT OBJECT TO SEND OUR QUERY TO DATA BASE AND TO BRING DATABASE RESULT

CONNECTION- FROM JAVA APPLITION TO DATABASE SOME NETWORK CONNECTION IS REQUIREMENT

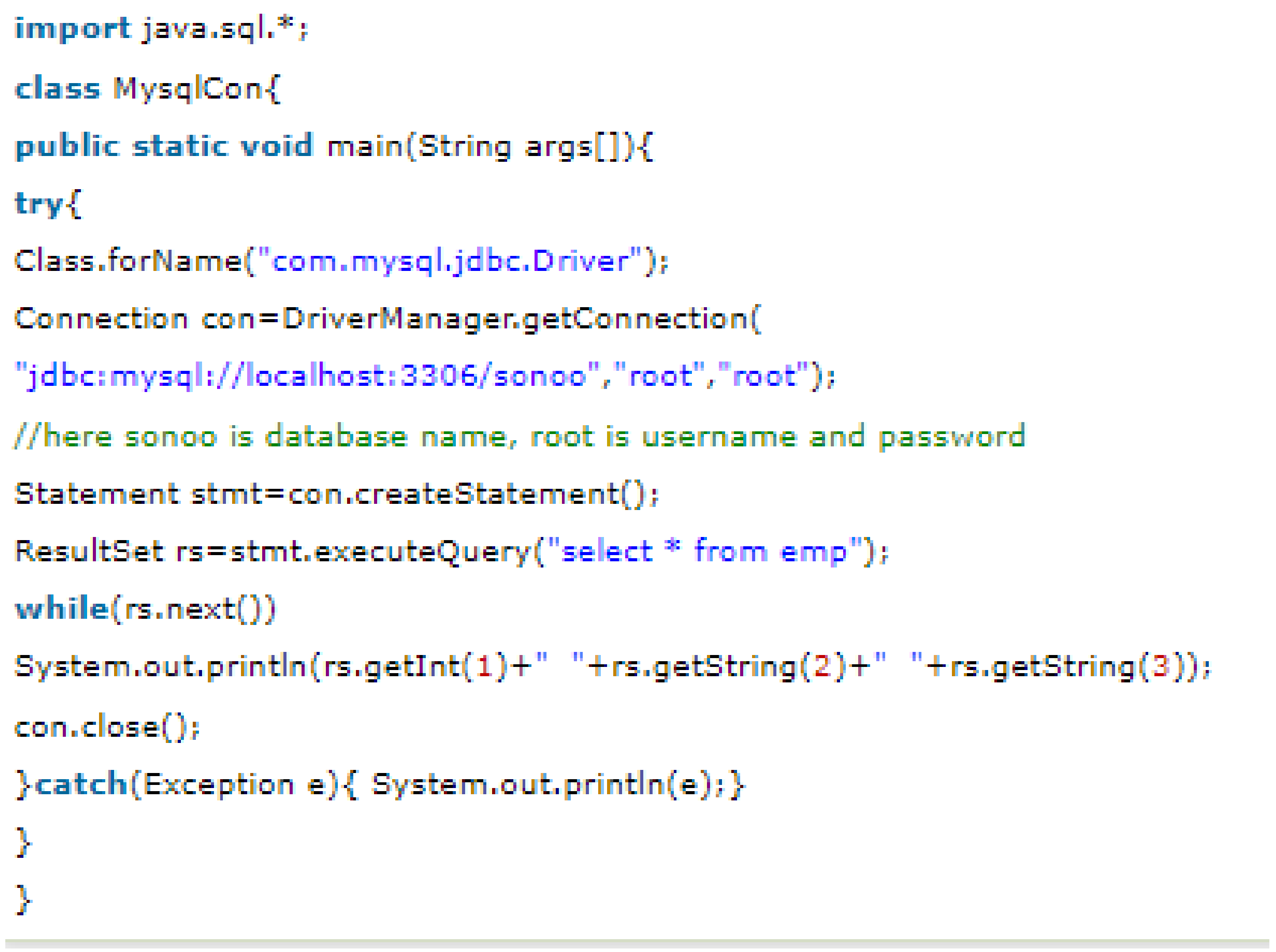
DATA BASE DRIVER IS A TRANSLATOR THAT CONVERT JAVA CALLS INTO DATA BASE CALL and vice vera

Creating JDBC Application

There are following six steps involved in building a JDBC application –

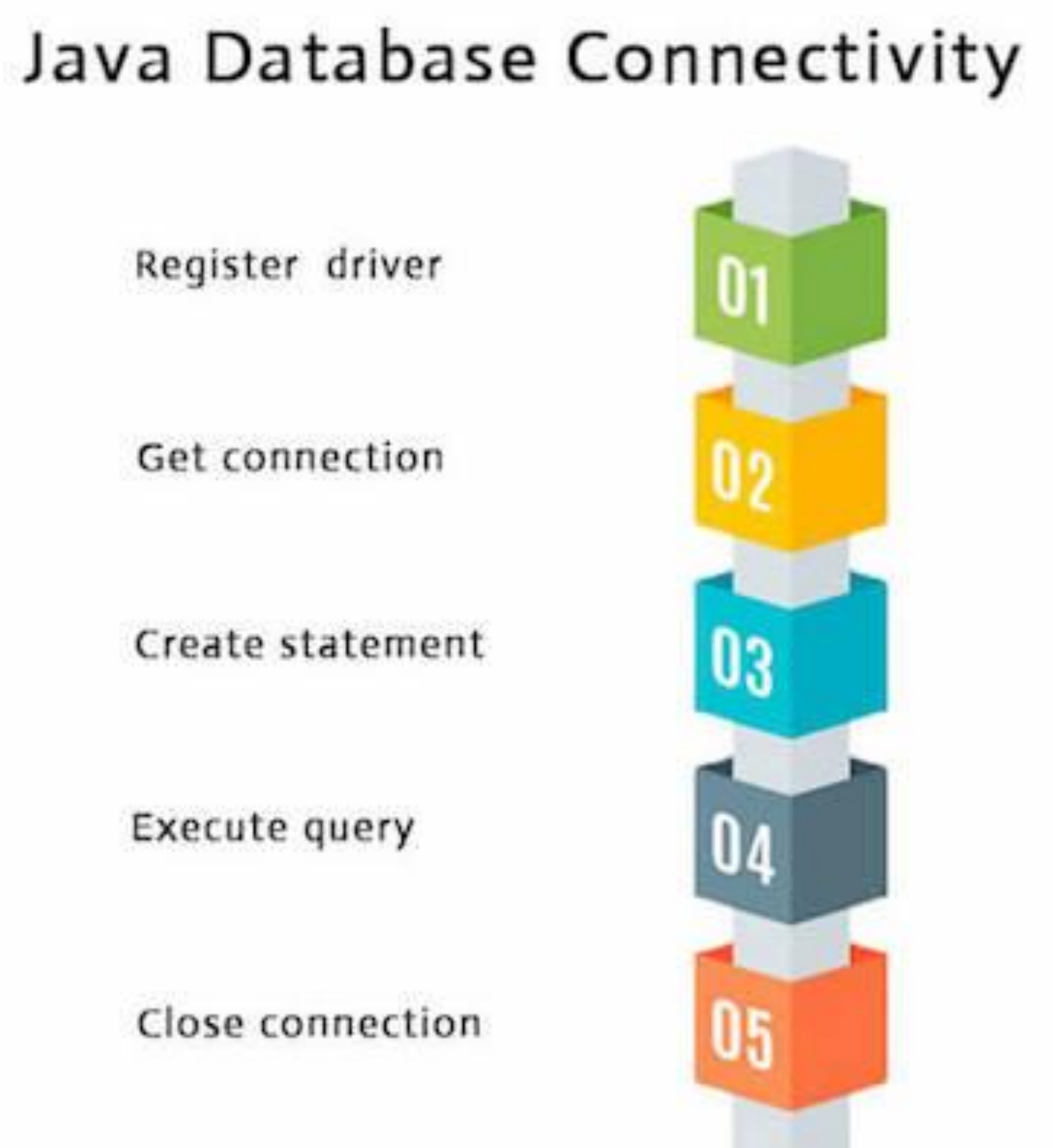
1. Import the packages: Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using import java.sql.* will suffice.
2. Register the JDBC driver: Requires that you initialize a driver so you can open a communication channel with the database.
3. Open a connection: Requires using the DriverManager.getConnection() method to create a Connection object, which represents a physical connection with the database.
4. Execute a query: Requires using an object of type Statement for building and submitting an SQL statement to the database.
5. Extract data from result set: Requires that you use the appropriate ResultSet.getXXX() method to retrieve the data from the result set.
6. Clean up the environment: Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

Example of creating JDBC Application:: overview



```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

Java Database Connectivity with 5 Steps



Import JDBC Packages

- import java.sql.* ; // for standard JDBC programs
- import java.math.* ; // for BigDecimal and BigInteger support

1) Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

```
public static void forName(String className)throws ClassNotFoundException
```

Example to register the OracleDriver class

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2) Create the connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

1) public static Connection getConnection(String url)throws SQLException
2) public static Connection getConnection(String url,String name,String password) throws SQLException

Example to establish connection with the Oracle database
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","password");

RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	**jdbc:mysql:**//hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@**hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2:**hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds:**hostname: port Number/databaseName

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

public Statement createStatement()throws SQLException

Commonly used methods of Statement interface:

Statement stmt=conn.createStatement();

There are 3 types of Statements, as given below:

Statement: It can be used for general-purpose access to the database. It is useful when you are using static SQL statements at runtime.

PreparedStatement: It can be used when you plan to use the same SQL statement many times. The PreparedStatement interface accepts input parameters at runtime.

CallableStatement: CallableStatement can be used when you want to access database stored procedures.

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

```
public ResultSet executeQuery(String sql)throws SQLException
```

Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");


while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

- **boolean execute (String SQL): Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.**
- **int executeUpdate (String SQL): Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.**
- **ResultSet executeQuery (String SQL): Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.**

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

```
public void close()throws SQLException
```

Example to close connection

```
con.close();
```

Java Database Connectivity with Oracle

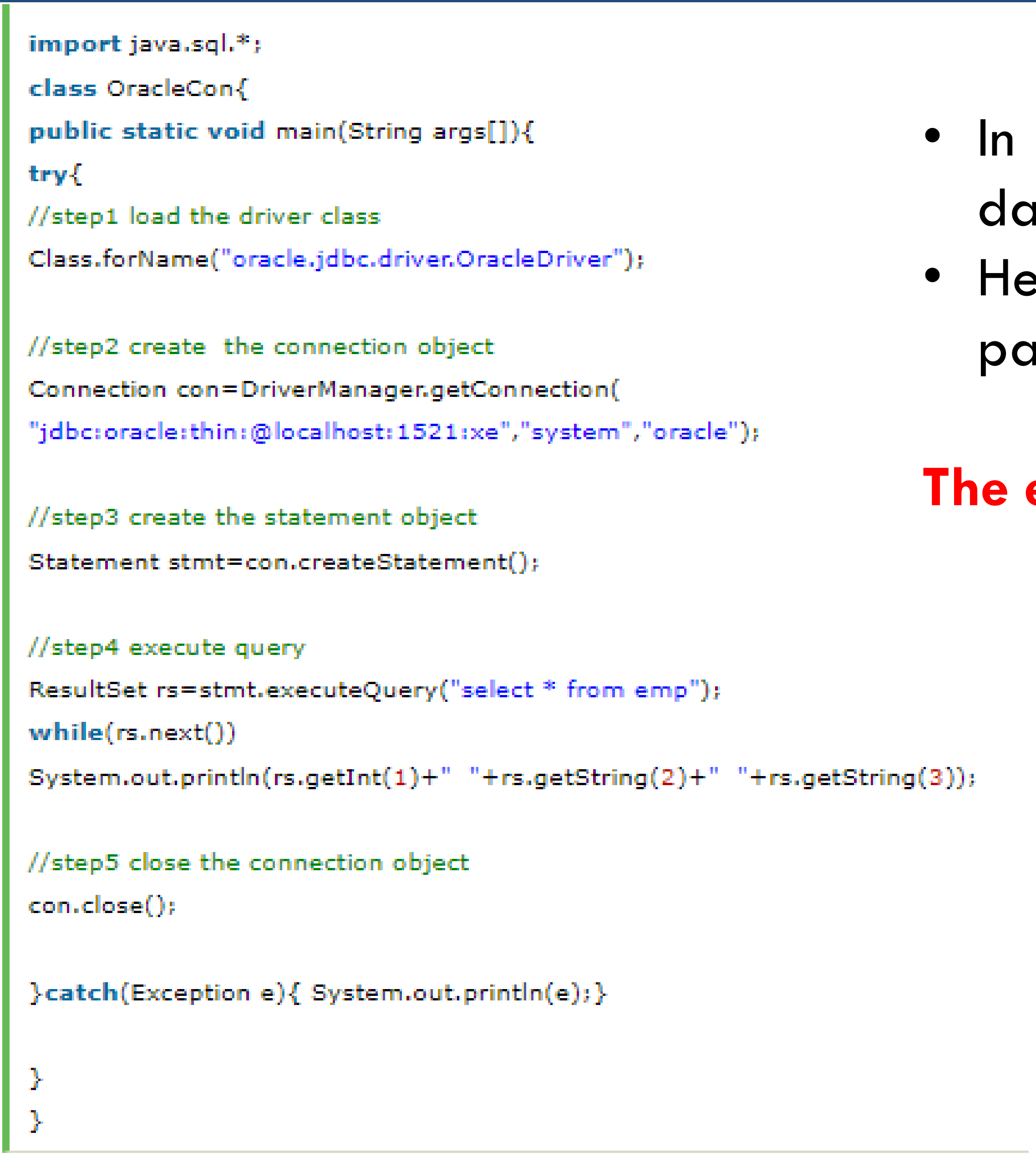
- To connect java application with the oracle database, we need to follow 5 following steps. In this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:**Driver class:** The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver.**
- **Connection URL:** The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file.
- **Username:** The default username for the oracle database is **system.**
- **Password:** It is the password given by the user at the time of installing the oracle database.

Create a Table

Before establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table.

```
create table emp(id number(10),name varchar2(40),age number(3));
```

Example to Connect Java Application with Oracle database



```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();

//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

//step5 close the connection object
con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

- In this example, we are connecting to an Oracle database and getting data from **emp** table.
- Here, **system** and **oracle** are the username and password of the Oracle database.

The example will fetch all the records of emp table.

Note: To connect java application with the Oracle database ojdbc14.jar file is required to be loaded.

Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.

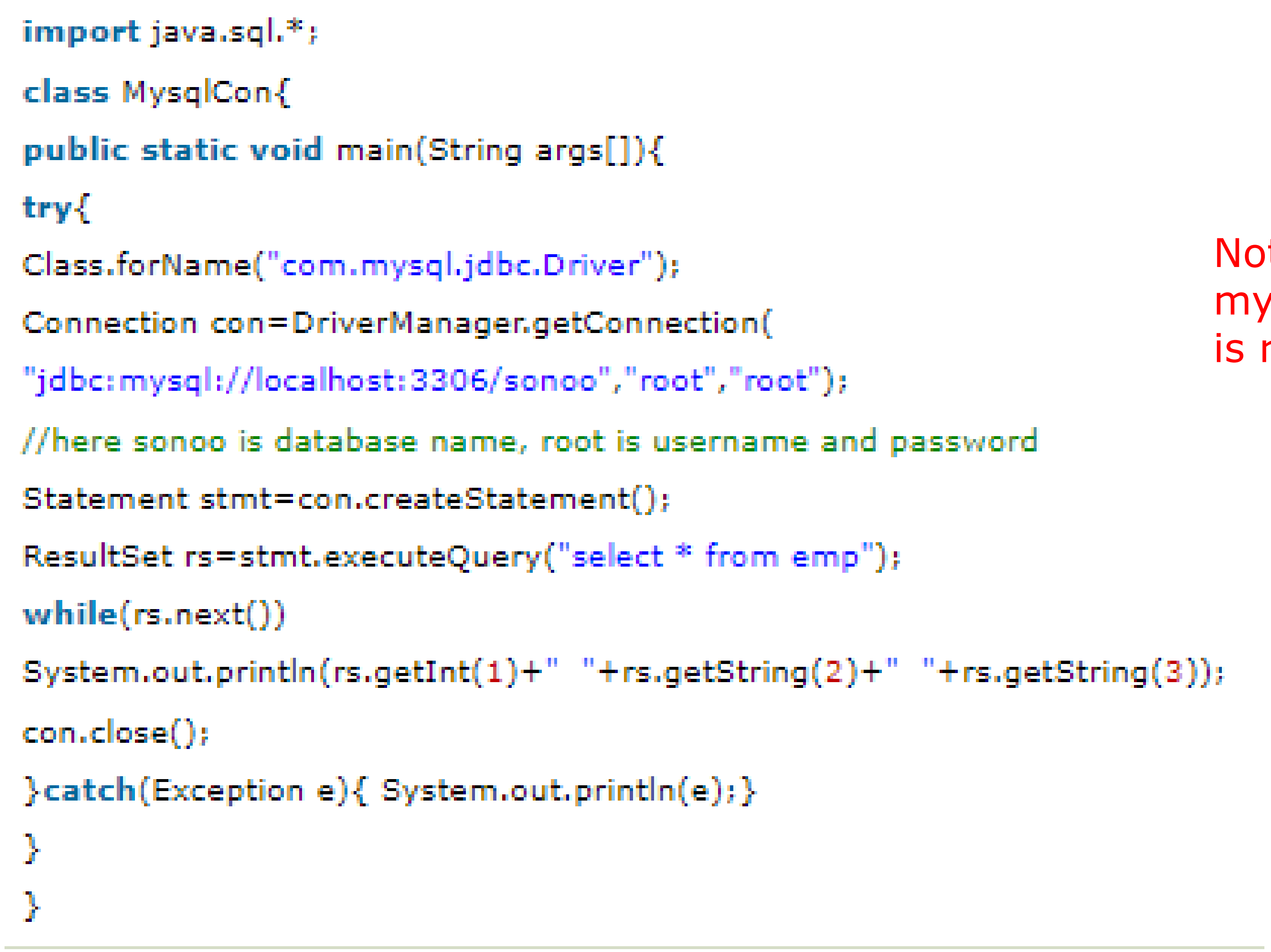
In this example we are using MySql as the database. So we need to know following information for the mysql database:

1. Driver class: The driver class for the mysql database is com.mysql.jdbc.Driver.
2. Connection URL: The connection URL for the mysql database is jdbc:mysql://localhost:3306/sonoo where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. Username: The default username for the mysql database is root.
4. Password: It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

```
create database sonoo;
use sonoo;
create table emp(id int(10),name varchar(40),age int(3));
```

Example to Connect Java Application with mysql database



```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

Note: To connect java application with the mysql database, **mysqlconnector.jar** file is required to be loaded.

DriverManager class

- The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Method	Description
1) public static void registerDriver(Driver driver):	is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver):	is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection getConnection(String url):	is used to establish the connection with the specified url.
4) public static Connection getConnection(String url,String userName,String password):	is used to establish the connection with the specified url, username and password.

Connection interface

Common methods are:

1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.
2) public Statement createStatement(int resultSetType,int resultSetConcurrency): Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
3) public void setAutoCommit(boolean status): is used to set the commit status.By default it is true.
4) public void commit(): saves the changes made since the previous commit/rollback permanent.
5) public void rollback(): Drops all changes made since the previous commit/rollback.
6) public void close(): closes the connection and Releases a JDBC resources immediately.

- A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

The important methods of Statement interface are as follows:

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.

2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) public boolean execute(String sql): is used to execute queries that may return multiple results.

4) public int[] executeBatch(): is used to execute batch of commands.

ResultSet interface

- The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

- Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATABLE);

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String

Prepared Statement interface

- The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.
- Let's see the example of parameterized query:

String sql="insert into emp values(?,?,?)";

- As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.
- Why use PreparedStatement?
- **Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

THANKYOU