Introduction to SQL

- SQL stands for *Structured Query Language*. It is a kind of language(mostly like English).
- SQL is undoubtedly the most popular and widely-used open source Database Management System (DBMS) language.
- It is simple to set up and use.
- SQL is mainly used to create a table, enter data into table, update records in a table and retrieving data from a table.

MySQL

MySQL is one of the most popular Database Management System (DBMS).

- MySQL is a database system used on the web.
- MySQL is a database system that runs on a server.
- MySQL is ideal for both small and large applications.
- MySQL is very fast, reliable, and easy to use.
- MySQL uses standard **SQL.**
- MySQL works on a number of platforms.
- MySQL is free to download and use.
- MySQL is developed, distributed, and supported by Oracle Corporation.
- MySQL is named after co-founder Monty Widenius's daughter: My

Why install MySQL locally?

- Installing MySQL on a simple PC allows a user to safely create and test a web application without affecting the data or systems on the actual database management systems.
- Let's discuss the installation information for MySQL on Windows.

All-in-One packages

There are some excellent all-in-one Windows distributions that contain Apache, PHP (Hypertext Pre-Processor), MySQL and other applications in a single installation file, e.g. XAMPP, WAMP Server, etc.

https://www.apachefriends.org/download.html
XAMPP for Windows

http://www.wampserver.com/en/
WAMP for Windows

Manual installation : Download MySQL

1. Go to the link: dev.mysql.com/downloads/
2. Open MySQL Community Server
3. Download the "Without installer" (Archive) version.

Direct Link: https://dev.mysql.com/get/Downloads/MySQL-8.0/mysql-8.0.13-winx64.zip

Manual installation : Extract files

- Let us install MySQL to C:mysql.
  - Extract the files in ZIP to C: drive.
  - Rename the folder mysql-x.x.xx-win32 to mysql.

Note:

- MySQL can be installed anywhere in your system.
- If you want a lightweight installation, you can remove every sub-folder except for bin, data, scripts and share.

Manual installation : Move 'data' folder

- Place the data folder on another drive (or partition your drive) to make backups and re-installation easier.
  - For this purpose, create a folder, say D://data/.
  - Move the contents of C://mysql/data into it.

- Now, you have two folders, C://mysql/data and D://data/.
  - The original C://mysql/data folder can be removed now.

Manual installation : File configuration

MySQL provides several configuration methods but, in general, it is easier to create a `my.ini` file in the mysql folder.

The simplest `my.ini` file is:

```
[mysqld]
# installation directory
basedir="C:\mysql\"

# data directory
datadir="D:\data\"
```

Note:
Remember to change these folder locations if you have installed MySQL or the data folder elsewhere.

Manual installation : Test installation

- The MySQL server now should be started by running C:mysql/bin/mysqld.exe.

Open a command box (Start > Run > cmd) and enter the following commands:

```
mysqld --console --initialize
```
(Remember the password generated)

- This will start the MySQL server which listens for requests on localhost port 3306 with the random generated password. You can now start the MySQL command line tool and connect to the database.

Open another command box and enter and use the random password to login:

```
mysql -u root -p
```

Note: This will show a welcome message and the mysql> prompt.

Manual installation : Windows service

- The easiest way to start MySQL is to add it as a Windows service. From a command prompt, enter:

```
cd mysqlbin
mysqld --install
```

Open the

1. Control Panel,
2. Administrative Tools, then
3. Services and
4. Double-click MySQL.
5. Set the Startup type to "Automatic" to ensure MySQL starts every time you boot your PC.

Basic Statements

- DISPLAY DATABASES

```
show databases;
```

- CREATE DATABASE

```
create database test;
```

- SET CURRENT DATABASE (test)

```
use test;
```

SHOW statement shows the databases already present.

CREATE statement is used to create a new database.

USE is used to set the current working database.

Statement : CREATE

To begin with, the table creation command requires the following details –

- Name of the table
- Name of the fields
- Definitions for each field

Syntax

```
CREATE TABLE table_name (
        column_name1 column_type1,
        column_name2 column_type2 …);
```

➢ Create a table, say `JavaCourse` under `test` database

```
create table JavaCourse(
        Roll Integer primary key,
        Name Varchar(30),
        Marks Integer not null,
        Grade Varchar(2));
```

Statement : DESC

In practice, you use the DESC statement which is a shorthand of the DESCRIBE statement. These statements are used to view the structure, datatype, keys and constraints used in a table.

Syntax

```
DESC table_name;
```

➢Describe table 'JavaCourse'

```
desc JavaCourse;
```

Field	Type	Null	Key	Default
Roll	int(11)	NO	PRI	*NULL*
Name	varchar(30)	YES		*NULL*
Marks	int(11)	NO		*NULL*
Grade	varchar(2)	YES		*NULL*

Statement : INSERT

You can insert data into the existing MySQL table by using the mysql> prompt or by using any script like PHP or any programming language using proper drivers using the **INSERT** statement.

Syntax

```
INSERT INTO table_name VALUES(Value1,Value2,…, Value n);
```

- INSERT

```
insert into JavaCourse values (01,'Debasish', 75, 'A');
insert into JavaCourse values(02,'Nilanjan', 85, 'EX');
insert into JavaCourse values(03,'Tauheed', 65, 'B');
insert into JavaCourse values(04,'Priyabrata', 78, 'A')
```

Roll	Name	Marks	Grade
1	Debasish	75	A
2	Nilanjan	85	EX
3	Tauheed	65	B
4	Priyabrata	78	A

Statement : UPDATE

UPDATE statement is used to modify previously inserted data in a table.

Syntax

```
UPDATE table_name SET
        ColName1=Value1,
        ColName2=Value2,
        ,…,
        WHERE ColName1=Value1);
```

- UPDATE

```
update JavaCourse set Name='Debasis' where Name='Debasish';
update JavaCourse set Marks=85, Grade='Ex' where Name='Debasis';
```

Roll	Name	Marks	Grade
1	Debasis	85	Ex
2	Nilanjan	85	EX
3	Tauheed	65	B
4	Priyabrata	78	A

Statement : SELECT

SELECT statement is used to retrieve data from a table.

Syntax

```
SELECT <ColName1> <ColName2>…<ColNameP>
FROM table_name
WHERE <Condition>;
```

Roll	Name	Marks	Grade
1	Debasis	85	Ex
2	Nilanjan	85	EX
3	Tauheed	65	B
4	Priyabrata	78	A

name	roll
Debasis	1
Nilanjan	2
Tauheed	3
Priyabrata	4

```
select * from JavaCourse;

select name, roll from JavaCourse;

select * from JavaCourse where marks>80;

select * from JavaCourse where name like '%jee%';

select * from JavaCourse order by marks;
```

Roll	Name	Marks	Grade
1	Debasis	85	Ex
2	Nilanjan	85	EX

Roll	Name	Marks	Grade

Roll	Name	Marks	Grade
3	Tauheed	65	B
4	Priyabrata	78	A
1	Debasis	85	Ex
2	Nilanjan	85	EX

Statement : DELETE

DELETE FROM statement is used to delete a record from any MySQL table.

Syntax

```
DELETE FROM table_name
WHERE <Condition>;
```

Roll	Name	Marks	Grade
1	Debasis	85	Ex
2	Nilanjan	85	EX
4	Priyabrata	78	A

- DELETE

```
delete from JavaCourse where marks <66;
```

Statement : DROP

DROP TABLE statement is used to drop an existing MySQL table, but you need to be very careful while deleting any existing table because the data lost will not be recovered after deleting a table.

Syntax

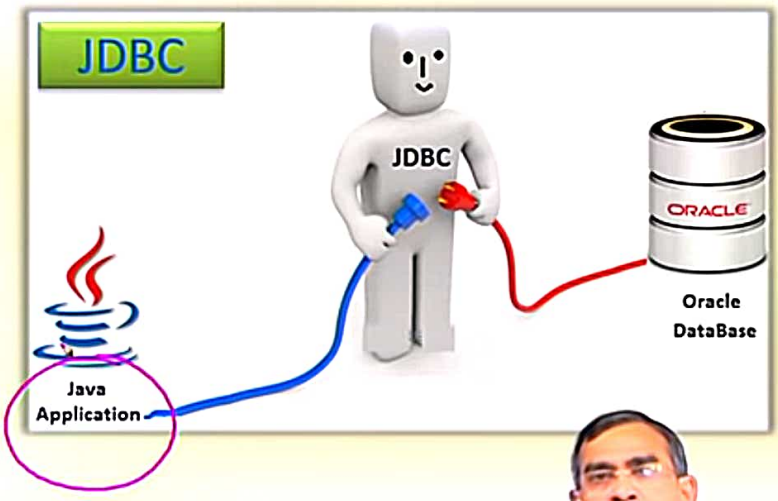
```
DROP TABLE table_name;
```

- DROP TABLE

```
drop table JavaCourse;
```


JDBC: Java Database Connectivity

- JDBC is a standard Java API for handling database related activities.
- In Java, there is a package java.sql having number of classes for database related programming.
- It includes java.sql.DriverManager class and two interfaces java.sql.Driver and java.sql.Connection.



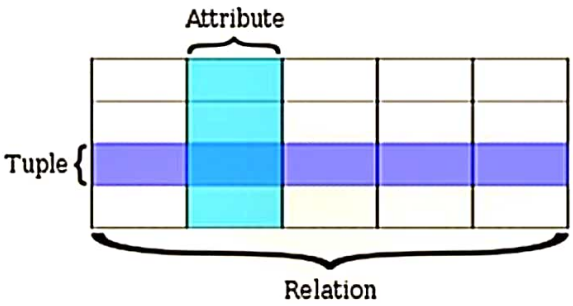
JDBC: Characteristics

- JDBC is a SQL (Structured Query Language)-level API, which is very popular for RDBMS (Relational Database Management System).
- It is compatible with the most of the popular database management systems, namely OracleDB, MySQL, Sybase, Microsoft SQL, etc.
- It is simple and easy to implement.

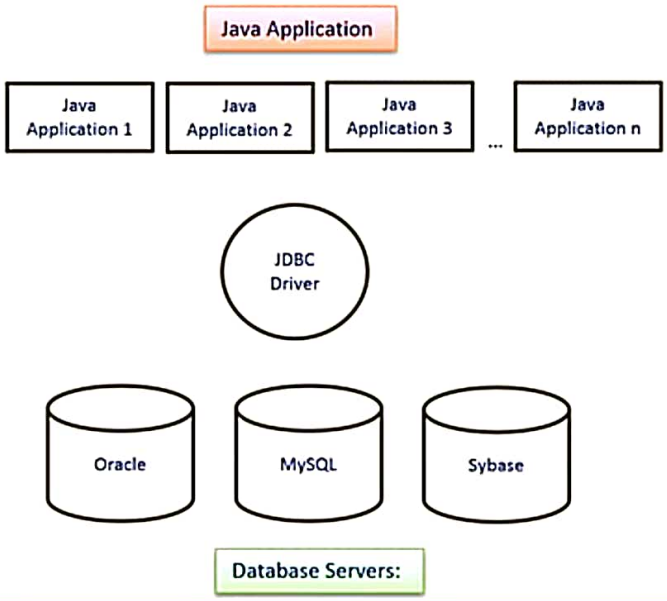
Relational database

- A relational database is a database that allows for queries which typically use Structured Query Language (SQL) to store and retrieve data.
- A relational database stores information by means of tables. A table is referred to as a relation in the sense that it is a collection of objects of the same type (rows).
- Examples:

  MS SQL Server, IBM DB2, Oracle, MySQL , etc.



JDBC: Characteristics



- In this course, we shall refer to JDBC to connect to MySQL database server.
- The approach is same to any other database.

Note:

- The JDBC driver for different database is different. But, as an end-user, we don't have to bother about their implementation.

Why JDBC?

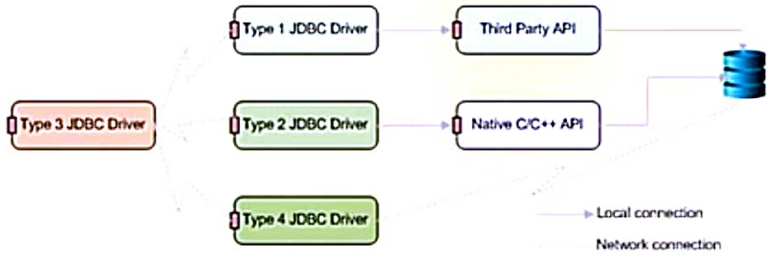
- Write once, run anywhere
  - Multiple client and server platforms.
- Object-relational mapping
  - Databases optimized for searching/indexing.
  - Objects optimized for engineering/flexibility.
- Network independence
  - Works across Internet Protocol.
- Database independence
  - Java can access any database vendor.

JDBC: Structure

- JDBC is a SQL-level API. It means that the JDBC allows to construct SQL statements and embed them inside Java API calls.
- The JDBC API is an implementation to interact a particular database engine. This implementation is called JDBC Driver.

JDBC: Types

JDBC Driver is a software component that enables Java application to interact with the database.



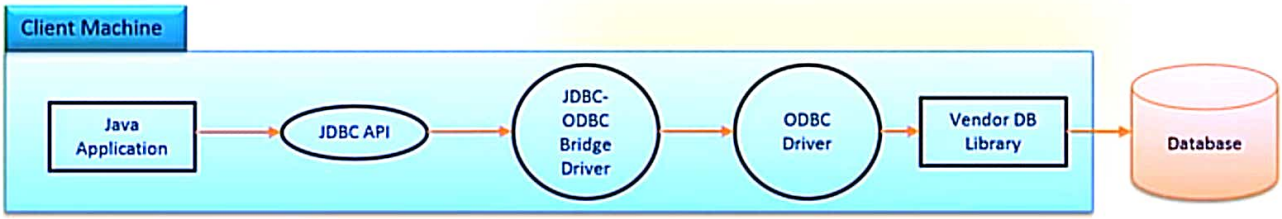
There are 4 types of JDBC drivers:

- Type – 1 : JDBC-ODBC bridge driver
- Type – 2 : Native-API driver (partially Java driver)
- Type – 3 : Network Protocol driver (fully Java driver)
- Type – 4 : Thin driver (fully Java driver)

JDBC: Type 1 – JDBC-ODBC

Type – I : JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.



Note: Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC B

JDBC: Type 1 – JDBC-ODBC

Type – I : JDBC-ODBC bridge driver

Advantages:

1. Easy to use.
2. Can be easily connected to any database.

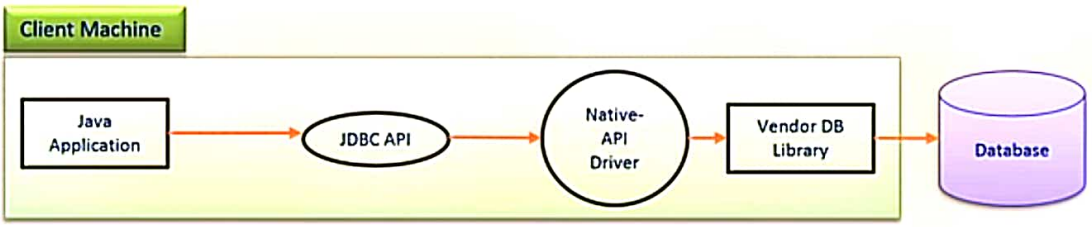
Disadvantages:

1. Performance degraded because JDBC method call is converted into the ODBC function calls.
2. The ODBC driver needs to be installed on the client machine.

JDBC: Type 2 – Native-API

Type – II : Native-API driver (partially Java driver)

The native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.



JDBC: Type 2 – Native-API

Type – II : Native-API driver (partially Java driver)

Advantages:

1. Performance upgraded than JDBC-ODBC bridge driver.

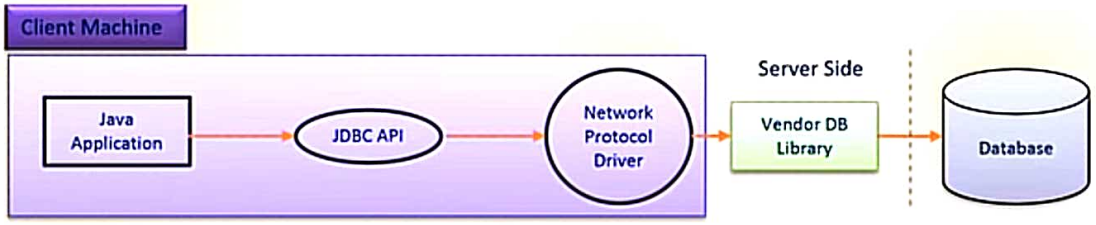
Disadvantages:

1. The native driver needs to be installed on the each client machine.
2. The vendor client library needs to be installed on client machine.

JDBC: Type 3 – Network Protocol

Type – III : Network Protocol driver (fully Java driver)

The network protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.



JDBC: Type 3 – Network Protocol

Type – III : Network Protocol driver (fully Java driver)

Advantages:

1. No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

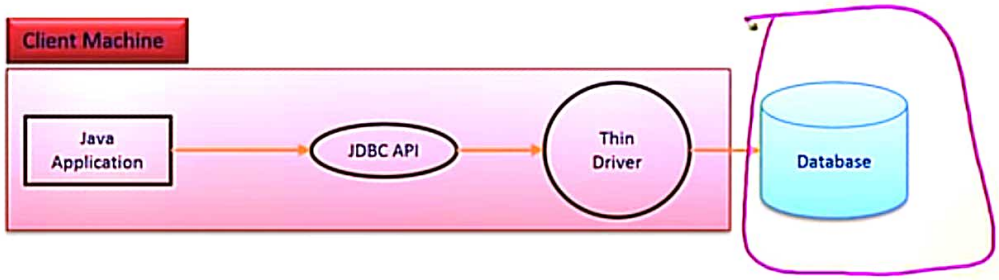
Disadvantages:

1. Network support is required on client machine.
2. Requires database-specific coding to be done in the middle tier.
3. Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

JDBC: Type 4 – Thin Driver

Type – IV : Thin driver (fully Java driver)

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.



JDBC: Type 4 – Thin Driver

Type – IV : Thin driver (fully Java driver)

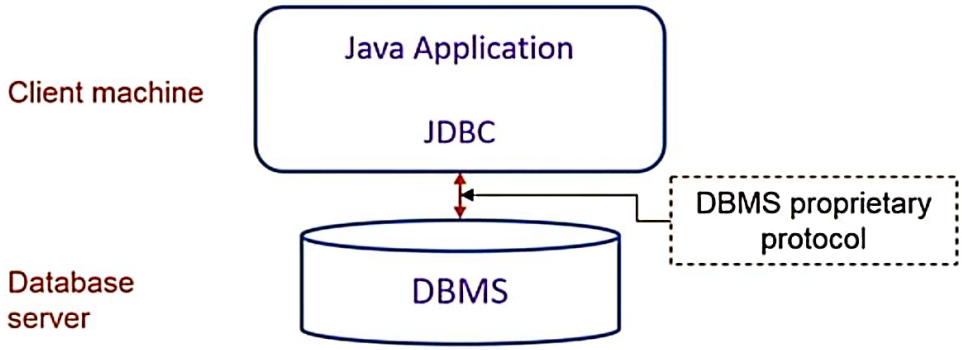
Advantages:

1. Better performance than all other drivers.
2. No software is required at client side or server side.

Disadvantages:

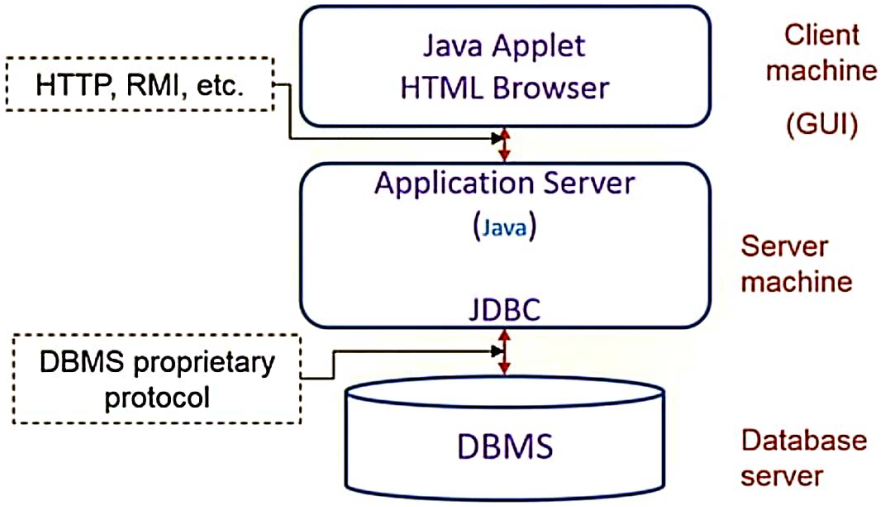
1. Drivers depend on the database.

JDBC Architecture : Two-tier



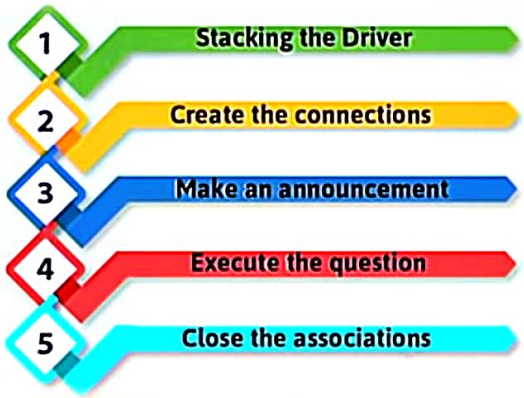
Two-tier architecture

JDBC Architecture : Three-tier



Three-tier architecture

JDBC steps



- Load a JDBC driver.
- Create connections.
- Connect to the data source.
- Execute SQL statement(s).
- Map the results to data structures.

JDBC driver

- Acts as the gateway to a database.
- Not actually a Windows "*driver*", it is just a .jar file.
- For MySQL JDBC driver:
  mysql-connector-java-8.0.12-bin.jar (current)

MySQL Driver Link
https://dev.mysql.com/downloads/connector/j/8.0.html

Common JDBC components

Class/Interface	Description
`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 sub-protocol 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. In stead, 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`	This **class** retrieves data 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.

Loading JDBC driver

- For MySql:

```
Class.forName ("com.mysql.cj.jdbc.Driver").newInstance();
```

- For Oracle:

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

- For MS SQL Server :

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
```

Establishing connection

- Create a `Connection` object
- Use the `DriverManager` to grab a connection with the `getConnection()` method

```
Connection conn = null;
String userName = "guest";
String password = "guest";
String url = "jdbc:mysql://10.14.100.141/test";
conn = DriverManager.getConnection (url, userName, password);
```

Three types of statements

Types	Description
`Statement`	For executing a simple SQL statement
`PreparedStatement`	For executing a precompiled SQL statement
`CallableStatement`	For executing a database stored procedure

Executing SQL statements

Create a statement object from the connection

```
Statement stmt = null;
stmt = conn.createStatement();

stmt.execute("show tables");
stmt.execute("insert into JavaCourse values (01,'Debasis', 85, 'Ex')");
```

Useful methods in Statement class

Methods	Description
`executeQuery()`	Executes SQL query and returns the data in a table (`ResultSet`) object. This method is used for SQL command that expects a return data from a database.
`executeUpdate()`	Used to execute INSERT, UPDATE, DELETE, CREATE TABLE, DROP TABLE, ALTER TABLE Returns the number of rows that are affected in the database
`execute()`	Generic method for executing simple statements, stored procedures, prepared statements. It can be used when the statement is either related to query or update. This method returns true *(if query yields a row)* otherwise returns false.
`getMaxRows()`	Determines the number of rows a `ResultSet` can contain.

PrepareStatement : An example

```
PreparedStatement pstmt = null;

String QryString = "INSERT INTO JavaCourse (Roll,Name,Marks,Grade) VALUES(?,?,?,?)";
pstmt = conn.prepareStatement(QryString);

pstmt.setInt(1, 12);
pstmt.setString(2, "ABC");
pstmt.setInt(3, 64);
pstmt.setString(4, "Ex");
pstmt.executeUpdate();
```

ResultSet

- A `ResultSet` provides access to a table of data generated by executing a Statement.
- Only one `ResultSet` per Statement can be open at once.
- The table rows are retrieved in sequence.
- A `ResultSet` maintains a cursor pointing to its current row of data.
- The `next()` method moves the cursor to the next row.
  - you can't rewind.

Useful ResultSet methods

Methods	Description
`boolean next()`	– attempts to move to the next row in ResultSet – the first call to next() positions cursor at the first row – returns false if there are no more rows
`Type getType(int columnIndex)`	– returns the given field as the given type – fields indexed starting at 1 (not 0)
`Type getType(String columnName)`	– same, but uses name of field – less efficient
`void close()`	– disposes of the ResultSet – allows you to re-use the Statement that created it
`int findColumn(String columnName)`	– looks up column index given column name

Matching with Java and SQL data types

SQL type	*Java class*	*ResultSet method*
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

Map the results to data structures

```
ResultSet rs = null;
String NameString, RollString, MarksString, GradeString;

stmt.execute("SELECT * FROM JavaCourse");
rs = stmt.getResultSet();

while (rs.next()) {
    NameString = rs.getString("Name");
    RollString = rs.getString("Roll");
    MarksString = rs.getString("Marks");
    GradeString = rs.getString("Grade");
}
```

JDBC exceptions

SQLException is an Exception class which provides information on database access errors.

```
try{
  ….

}

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

Connecting to MySQL server

The following class **Connect.java** is to connect from a Java application to MySQL database server.

```
import java.sql.*;
   public class Connect {
       public static void main (String[] args) {
          Connection conn = null;
           try
           {
               String userName = "guest";
               String password = "guest";
               String url = "jdbc:mysql://10.14.100.141/test";
               Class.forName ("com.mysql.jdbc.Driver").newInstance ();
               conn = DriverManager.getConnection (url, userName, password);
               System.out.println ("Database connection established");
           }
```

Continued to the next slide...

Connecting to MySQL server...

...Continued from the previous slide

```
            catch (Exception e) {
                System.err.println ("Cannot connect to database server");
            } finally {
                if (conn != null) {
                    try {
                        conn.close ();
                        System.out.println ("Database connection terminated");
                    } catch (Exception e) { /* ignore close errors */ }
                } // if
            } // finally
    } // main()
} // Class
```

Creating a table

The following class creates a database from a Java application in the MySQL database server.

```
import java.sql.*;

public class CreateTableJavaDataBase    {
       public static void main (String[] args)   {
           Connection conn = null;
           Statement stmt = null;
           ResultSet rs = null;
           String TableName;
           try  {
                String userName = "guest";
                String password = "guest";
                String url = "jdbc:mysql://10.14.100.141/test";
                Class.forName ("com.mysql.jdbc.Driver").newInstance();
                conn = DriverManager.getConnection (url, userName, password);
```

Continued to the n... e...

Creating a table...

...Continued from the previous slide

```
stmt = conn.createStatement();
stmt.execute("show tables");
rs = stmt.getResultSet();
System.out.println("Result before creating the table…");
while (rs.next()) {
    TableName = rs.getString("JavaCourse");  // To create a table having name JavaCourse
    System.out.println("Table Name: = "+TableName+"\n");
 }
    stmt.execute("create table JavaCourse(Roll Integer primary key, Name Varchar(30),
                  Marks Integer not null, Grade Varchar(2))");
```

Continued to the n...

Creating a table...

...Continued from the previous slide

```
stmt.execute("show tables");
          rs = stmt.getResultSet();
          System.out.println("Result after creating the table…\n");
          while (rs.next()) {
              TableName = rs.getString("MyTables");
              System.out.println("Table Name: = "+TableName+"\n");
          }
         } catch (SQLException ex){
              System.out.println("SQLException: " + ex.getMessage());
              System.out.println("SQLState: " + ex.getSQLState());
              System.out.println("VendorError: " + ex.getErrorCode());
            }
           catch (Exception e)     {
              System.err.println ("Cannot connect to database server");
          }
```

Continued to the n... e...

Creating a table...

```
        finally {
         if (rs != null) {
           try {
              rs.close();
           } catch (SQLException sqlEx) {//Ignore any code here… }
              rs = null;
         }
          if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException sqlEx) {// ignore code for this}
                stmt = null;
         }
          if (conn != null) {
          try {
                conn.close ();
              } catch (Exception e) { /* Ignore code for closing errors */ }
          }
       }
    } //main())
} // class
```

Inserting a record into table

```
import java.sql.*;

   public class InsertJavaDataBase
   {
       public static void main (String[] args)
       {
            Connection conn = null;
            Statement stmt = null;
            ResultSet rs = null;
            String NameString, RollString, MarksString, GradeString;
             try
             {
                 String userName = "guest";
                 String password = "guest";
                 String url = "jdbc:mysql://10.14.100.141/test";
                 Class.forName ("com.mysql.jdbc.Driver").newInstance();
                 conn = DriverManager.getConnection (url, userName, password);
```

Inserting a record into table...

```
stmt = conn.createStatement();
stmt.execute("SELECT * FROM JavaCourse");
rs = stmt.getResultSet();
System.out.println("\n\n ------- Result Before Insert---------\n");
while (rs.next()) {
        NameString = rs.getString("Name");
        RollString = rs.getString("Roll");
        MarksString = rs.getString("Marks");
        GradeString = rs.getString("Grade");
        System.out.println("Name: = "+NameString+"\t\t"+"Roll: = "+RollString+"\t\t"+"Marks: =
                           "+MarksString+" \t\t"+"Grade: = "+GradeString+"\n");
} //end while
stmt.execute("INSERT INTO JavaCourse values (01,'Debasish Kundu', 75, 'A')");
stmt.execute("INSERT INTO JavaCourse values(02,'Saikat Das', 85, 'EX')");
stmt.execute("INSERT INTO JavaCourse values(03,'Sandeep Banerjee', 65, 'B')");
stmt.execute("INSERT INTO JavaCourse values(04,'Raju Chatterjee', 78, 'A')");
stmt.execute("SELECT * FROM JavaCourse");
rs = stmt.getResultSet();
```

Inserting a record into table...

```
System.out.println("\n\n ------- Result After Insert---------\n");
while (rs.next()) {
       NameString = rs.getString("Name");
       RollString = rs.getString("Roll");
       MarksString = rs.getString("Marks");
       GradeString = rs.getString("Grade");
       System.out.println("Name: = "+NameString+"\t\t"+"Roll: = "+RollString +"\t\t"+"Marks: =
                                  "+MarksString+"\t\t"+"Grade: = "+GradeString+"\n");
     } //end while
  }
        catch (SQLException ex){
         // handle any errors
         System.out.println("SQLException: " + ex.getMessage());
         System.out.println("SQLState: " + ex.getSQLState());
         System.out.println("VendorError: " + ex.getErrorCode());
         }
         catch (Exception e)
       {
          System.err.println ("Cannot connect to database server");
       }
```

Inserting a record into table...

```
finally {
        if (rs != null) {
         try {
           rs.close();
         } catch (SQLException sqlEx) { } // ignore
           rs = null;
        }
        if (stmt != null) {
         try {
           stmt.close();
        } catch (SQLException sqlEx) { } // ignore
          stmt = null;
        }
        if (conn != null) {
          try {
                conn.close ();// System.out.println ("Database connection terminated");
              } catch (Exception e) { /* ignore close errors */ }
            }
         }
            }
    }
```

Inserting a record into table

```
import java.sql.*;
import java.util.Scanner;

   public class InsertJavaDataBase1
   {
       public static void main (String[] args)
       {
            Connection conn = null;
            PreparedStatement pstmt = null;
            Statement stmt = null;
            ResultSet rs = null;
            String Name, Grade, NameString, RollString, MarksString, GradeString; int Roll, Marks;

           try
           {
               String userName = "guest";
               String password = "guest";
               String url = "jdbc:mysql://10.14.100.141/test";
               Class.forName ("com.mysql.jdbc.Driver").newInstance();
               conn = DriverManager.getConnection (url, userName, password);
```

Inserting a record into table...

```
System.out.println("\n\n ------- Results before Insert ---------\n");
stmt = conn.createStatement();
stmt.execute("SELECT * FROM JavaCourse");
rs = stmt.getResultSet();
while (rs.next()) {
      NameString = rs.getString("Name");
      RollString = rs.getString("Roll");
      MarksString = rs.getString("Marks");
      GradeString = rs.getString("Grade");
      System.out.println("Name: = "+NameString+"\t\t"+"Roll: = "+RollString+"\t\t"+"Marks: =
                         "+MarksString+"\t\t"+"Grade: = "+GradeString+"\n");
      } //end while
      System.out.println("\n\n --- Input for the entries of table (JavaCourse) ---\n");
      Scanner in = new Scanner(System.in);
      System.out.println("\n Enter Name: \t");
      Name = in.nextLine();
      System.out.println("\n Enter Grade: \t");
      Grade = in.nextLine();
```

serting a record into table...

```
System.out.println("\n Enter Roll: \t");
Roll = in.nextInt();
System.out.println("\n Enter Marks: \t");
Marks = in.nextInt();
String QryString = "INSERT INTO JavaCourse (Roll,Name,Marks,Grade) VALUES(?,?,?,?)";
pstmt = conn.prepareStatement(QryString);
pstmt.setInt(1, Roll);
pstmt.setString(2, Name);
pstmt.setInt(3, Marks);
pstmt.seString(4, Grade);
pstmt.executeUpdate();
System.out.println("\n\n ------- Results after Insert ---------\n");
stmt.execute("SELECT * FROM JavaCourse");
rs = stmt.getResultSet();
while (rs.next()) {
   NameString = rs.getString("Name");
   RollString = rs.getString("Roll");
   MarksString = rs.getString("Marks");
   GradeString = rs.getString("Grade");
```

serting a record into table...

```
System.out.println("Name: = "+NameString+"\t\t"+"Roll: = "+RollString+"\t\t"+"Marks: =
                           "+MarksString+"\t\t"+"Grade: = "+GradeString+"\n");
  } //end while
}
  catch (SQLException ex){// handle any errors
       System.out.println("SQLException: " + ex.getMessage());
       System.out.println("SQLState: " + ex.getSQLState());
       System.out.println("VendorError: " + ex.getErrorCode());
      }
   catch (Exception e)
   {
     System.err.println ("Cannot connect to database server");
    }
   finally {
       if (rs != null) {
        try {
              rs.close();
            } catch (SQLException sqlEx) { } // ignore
           rs = null;
         }
```

Inserting a record into table...

```
if (stmt != null) {
            try {
              stmt.close();
           } catch (SQLException sqlEx) { } // ignore
             stmt = null;
         }
if (conn != null) {
            try {
                   conn.close ();// System.out.println ("Database connection terminated");
                }catch (Exception e) { /* ignore close errors */ }
               }
            }
if (pstmt != null) {
             try {
                   pstmt.close ();// System.out.println ("Database connection terminated");
                } catch (Exception e) { /* ignore close errors */ }
               }
            }
           }
```

Processing a table

```
import java.sql.*;
import java.sql.ResultSet;
public class ProcessJavaDataBase    {
       public static void main (String[] args)
       {
            Connection conn = null;
            Statement stmt = null;
            ResultSet rs = null;
            int TotalMarks = 0, Num_Student = 0;
            float Avg_Marks;
            String NameString, RollString, MarksString, GradeString;
            try {
                 String userName = "guest";
                 String password = "guest";
                 String url = "jdbc:mysql://10.14.100.141/test";
                 Class.forName ("com.mysql.jdbc.Driver").newInstance();
                 conn = DriverManager.getConnection (url, userName, password);
```

Processing a table...

```
  stmt = conn.createStatement();
  stmt.execute("SELECT * FROM JavaCourse");
  rs = stmt.getResultSet();
  System.out.println("\n\n ------- Results  ---------\n");
  while (rs.next()) {
       NameString = rs.getString("Name");
       RollString = rs.getString("Roll");
       MarksString = rs.getString("Marks");
       GradeString = rs.getString("Grade");
       TotalMarks = TotalMarks + Integer.parseInt(MarksString);
       System.out.println("Name: = "+NameString+"\t\t"+"Roll: = "+RollString+"\t\t"+"Marks:
                          = "+MarksString+"\t\t"+"Grade: = "+GradeString+"\n");
    } //end while
   rs.last();
   Num_Student = rs.getRow();
   Avg_Marks = TotalMarks / Num_Student;
   System.out.println("\n\n ------- AVERAGE Marks = "+Avg_Marks+"--------");
}
```

Updating a table

```
import java.sql.*;
import java.sql.ResultSet;
public class UpdateJavaDataBase
   {
       public static void main (String[] args)
       {
           Connection conn = null;
           Statement stmt = null;
           ResultSet rs = null;
           String NameString, RollString, MarksString, GradeString;
           try
            {
               String userName = "guest";
               String password = "guest";
               String url = "jdbc:mysql://10.14.100.141/test";
               Class.forName ("com.mysql.jdbc.Driver").newInstance();
               conn = DriverManager.getConnection (url, userName, password);
```

Updating a table...

```
stmt = conn.createStatement();
stmt.execute("SELECT * FROM JavaCourse");
rs = stmt.getResultSet();
System.out.println("\n\n ------- Result Before Update ---------\n");
while (rs.next()) {
     NameString = rs.getString("Name");
     RollString = rs.getString("Roll");
     MarksString = rs.getString("Marks");
     GradeString = rs.getString("Grade");
     System.out.println("Name: = "+NameString+"\t\t"+"Roll: = "+RollString+"\t\t"+"Marks: =
                        "+MarksString+"\t\t"+"Grade: = "+GradeString+"\n");
} //end while
// Update a row
 stmt.execute("update JavaCourse set Marks=85, Grade='Ex' where Name='Debasish Kundu'");
 stmt.execute("SELECT * FROM JavaCourse");
 rs = stmt.getResultSet();

 System.out.println("\n\n ------- Result After Update ---------\n");
```

Updating a table...

```
    while (rs.next()) {
       NameString = rs.getString("Name");
       RollString = rs.getString("Roll");
       MarksString = rs.getString("Marks");
       GradeString = rs.getString("Grade");
       System.out.println("Name: = "+NameString+"\t\t"+"Roll: = "+RollString+"\t\t"+"Marks: =
                          "+MarksString+"\t\t"+"Grade: = "+GradeString+"\n");
     } //end while
    }
   catch (SQLException ex){
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
   }
   catch (Exception e)
  {
    System.err.println ("Cannot connect to database server");
   }
```

Updating a table...

```
finally {
   if (rs != null) {
      try {
           rs.close();
       } catch (SQLException sqlEx) { } // ignore
        rs = null;
     }
     if (stmt != null) {
      try {
         stmt.close();
       } catch (SQLException sqlEx) { } // ignore
       stmt = null;
     }
     if (conn != null) {
       try {
          conn.close ();
           // System.out.println ("Database connection terminated");
           }                catch (Exception e) { /* ignore close errors */ }
          }
      } }
 }
```

Deleting a record

```
import java.sql.*;
import java.sql.ResultSet;
public class DeleteJavaDataBase    {
       public static void main (String[] args)
       {
           Connection conn = null;
           Statement stmt = null;
           ResultSet rs = null;
           String NameString, RollString, MarksString, GradeString;
           try
           {
               String userName = "guest";
               String password = "guest";
               String url = "jdbc:mysql://10.14.100.141/test";
               Class.forName ("com.mysql.jdbc.Driver").newInstance();
               conn = DriverManager.getConnection (url, userName, password);
               // System.out.println ("Database connection established");
```

Deleting a record...

```
System.out.println("\n\n ------- Result Before Delete ---------\n");
while (rs.next()) {
    NameString = rs.getString("Name");
    RollString = rs.getString("Roll");
    MarksString = rs.getString("Marks");
    GradeString = rs.getString("Grade");
    System.out.println("Name: = "+NameString+"\t\t"+"Roll: = "+RollString+"\t\t"+"Marks: =
                        "+MarksString+"\t\t"+"Grade: = "+GradeString+"\n");
 } //end while
// delete all rows
 stmt.execute("delete from JavaCourse");
 stmt.execute("SELECT * FROM JavaCourse");
 rs = stmt.getResultSet();
 System.out.println("\n\n ------- Result After Delete ---------\n");
while (rs.next()) {
     NameString = rs.getString("Name");
     RollString = rs.getString("Roll");
     MarksString = rs.getString("Marks");
     GradeString = rs.getString("Grade");
```


Deleting a record...

```
 System.out.println("Name: = "+NameString+"\t\t"+"Roll: = "+RollString+"\t\t"+"Marks: =
                            "+MarksString+"\t\t"+"Grade: = "+GradeString+"\n");
   } //end while
}
catch (SQLException ex){  // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
catch (Exception e){
    System.err.println ("Cannot connect to database server");
}
finally {
    if (rs != null) {
       try {
         rs.close();
       } catch (SQLException sqlEx) { } // ignore
         rs = null;
    }
```

Deleting a table...

```
import java.sql.*;
if (stmt != null) stmt.close();
public class DeleteTableJavaDataBase    {
        public static void main (String[] args)
        {} catch (SQLException sqlEx) { } // ignore
            conn = null;
    }       Statement stmt = null;
    if (conn != null) {
         try {String TableName;
              conn.close (); // System.out.println ("Database connection terminated");
         }  try
          catch (Exception e) { /* ignore close errors */ }
     }              String userName = "guest";
   }                String password = "guest";
  }                 String url = "jdbc:mysql://10.14.100.141/test";
}                   Class.forName ("com.mysql.jdbc.Driver").newInstance();
                    conn = DriverManager.getConnection (url, userName, password);
```

Deleting a record...

```
    System.out.println("Name: = "+NameString+"\t\t"+"Roll: = "+RollString+"\t\t"+"Marks: =
                                "+MarksString+"\t\t"+"Grade: = "+GradeString+"\n");
      } //end while
   }
   catch (SQLException ex){  // handle any errors
       System.out.println("SQLException: " + ex.getMessage());
       System.out.println("SQLState: " + ex.getSQLState());
       System.out.println("VendorError: " + ex.getErrorCode());
   }
   catch (Exception e){
       System.err.println ("Cannot connect to database server");
   }
   finally {
       if (rs != null) {
          try {
            rs.close();
          } catch (SQLException sqlEx) { } // ignore
            rs = null;
       }
```

Deleting a record...

```
    if (stmt != null) {
        try {
             stmt.close();
        } catch (SQLException sqlEx) { } // ignore
            stmt = null;
    }
    if (conn != null) {
        try {
             conn.close ();// System.out.println ("Database connection terminated");
        }
         catch (Exception e) { /* ignore close errors */ }
     }
   }
  }
}
```

Screenshot captured.

Deleting a table

```
import java.sql.*;
import java.sql.ResultSet;
public class DeleteTableJavaDataBase    {
       public static void main (String[] args)
       {
           Connection conn = null;
           Statement stmt = null;
           ResultSet rs = null;
           String TableName;

           try
            {
                 String userName = "guest";
                 String password = "guest";
                 String url = "jdbc:mysql://10.14.100.141/test";
                 Class.forName ("com.mysql.jdbc.Driver").newInstance();
                 conn = DriverManager.getConnection (url, userName, password);
```

Deleting a table

```
    stmt = conn.createStatement();
    stmt.execute("show tables");
    rs = stmt.getResultSet();
    System.out.println("\n\n ------- Result Before Delete Table ---------\n");
    while (rs.next()) {
           TableName = rs.getString("Tables in test"); // change this field name
           System.out.println("Table Name: = "+TableName+"\n");
      } //end while
     // Deleting a table called JavaCourse.
    stmt.execute("drop table JavaCourse");
    stmt.execute("show tables");
    rs = stmt.getResultSet();
    System.out.println("\n\n ------- Result After Delete Table ---------\n");
    while (rs.next()) {
          TableName = rs.getString("Tables in test");
          System.out.println("Table Name: = "+TableName+"\n");
      } //end while
  }
```

Deleting a table

```
catch (SQLException ex){
       // handle any errors
         System.out.println("SQLException: " + ex.getMessage());
         System.out.println("SQLState: " + ex.getSQLState());
         System.out.println("VendorError: " + ex.getErrorCode());
    }
 catch (Exception e)
 {
         System.err.println ("Cannot connect to database server");
  }
  finally {
      if (rs != null) {
           try {
             rs.close();
           } catch (SQLException sqlEx) { } // ignore
             rs = null;
          }
```

Deleting a table

```
    if (stmt != null) {
            try {
              stmt.close();
           } catch (SQLException sqlEx) { } // ignore
             stmt = null;
       }
    If (conn != null) {
            try {
                  conn.close ();
                  // System.out.println ("Database connection terminated");
                }
             catch (Exception e) { /* ignore close errors */ }
     }
   }
}
```