Marwadi education foundation

01CE0411 - 4 Credits

Unit #1

Java Database Connectivity

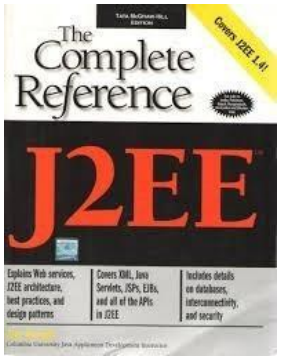
Prepared By
Prof. Ravikumar R Natarajan
Assistant Professor, CE Dept.

KNOWLEDGE IS THE CURRENCY
FOR THE 21st CENTURY

JDBC

Marwadi education foundation

Reference Book:
Complete Reference J2EE by James Keogh mcgraw publication
Chapter : 6 and 7



Web Reference:
www.javatpoint.com
www.tutorialspoint.com

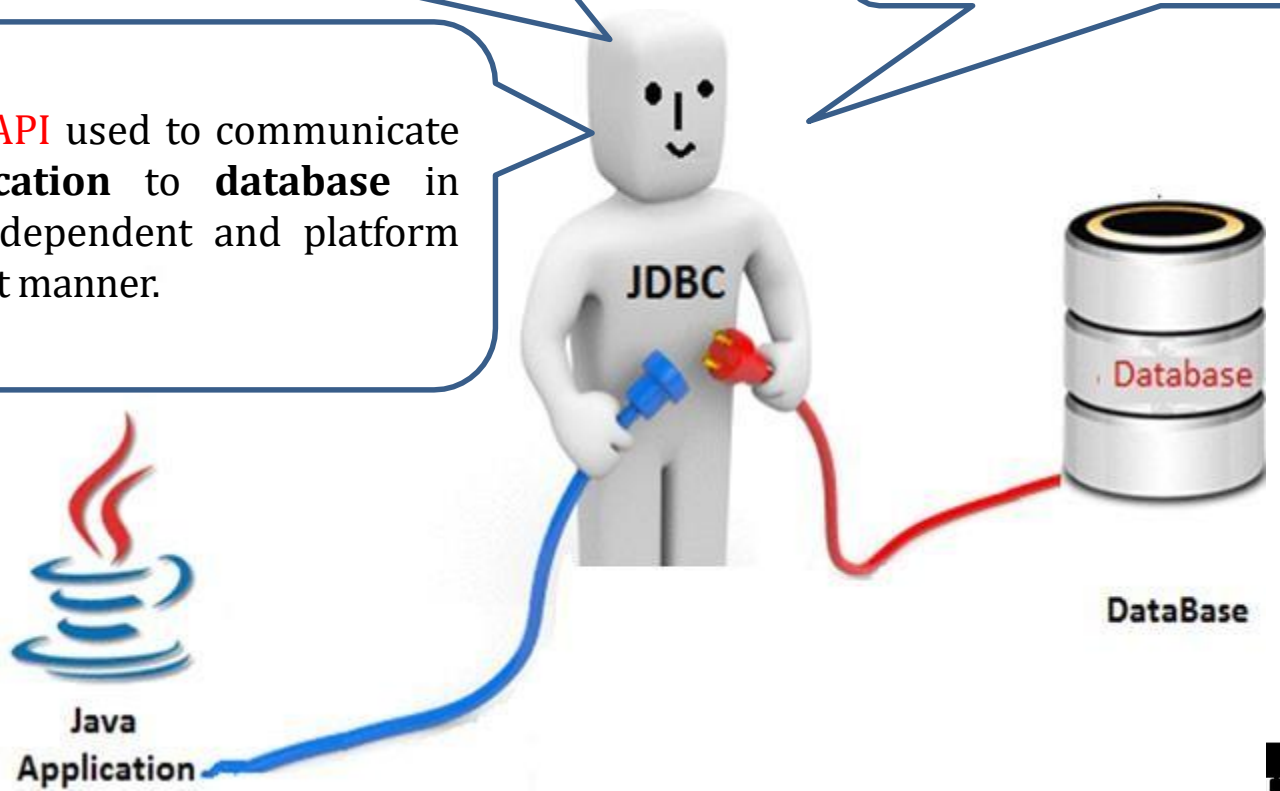
Contents

JDBC

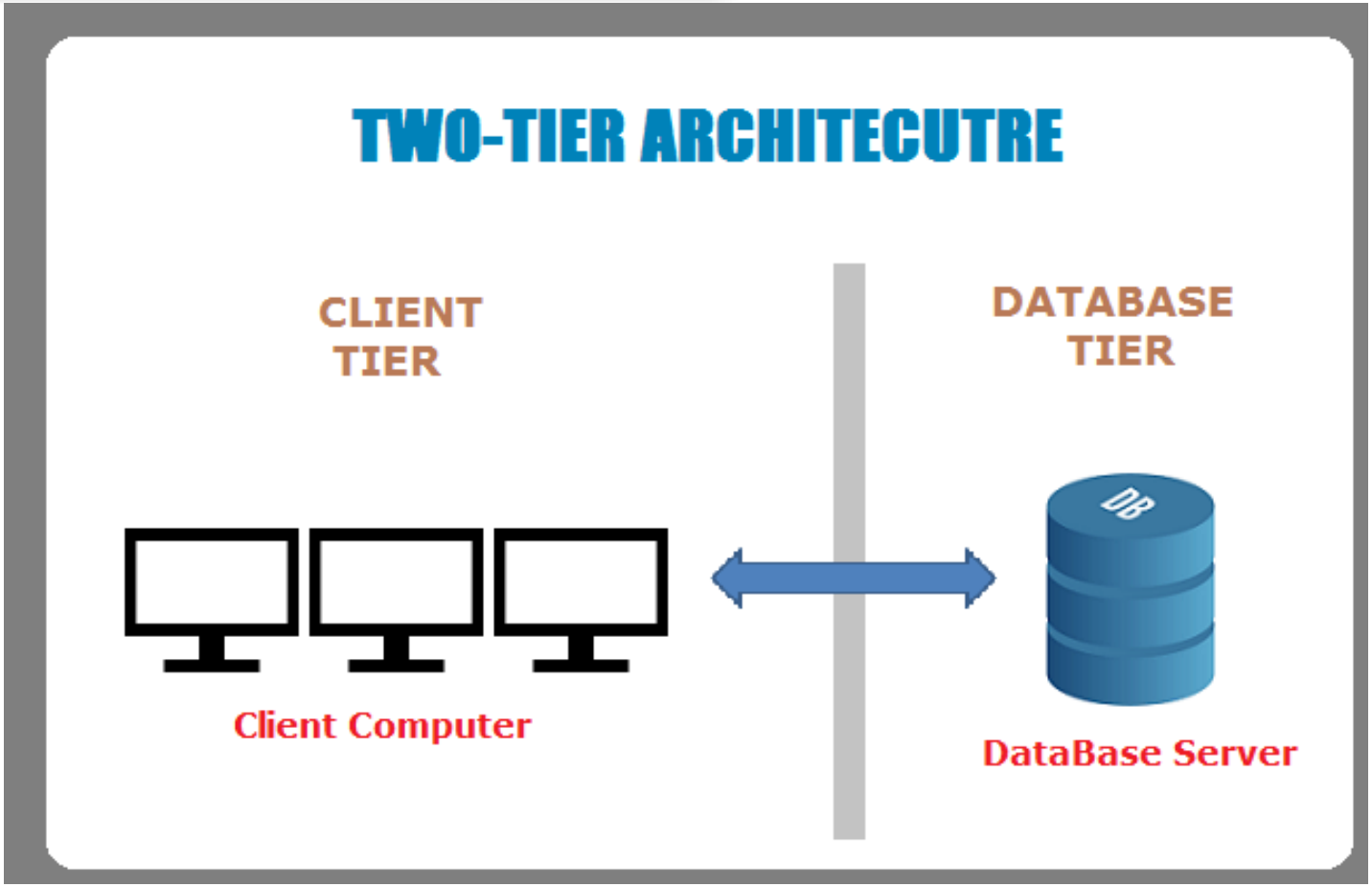
JDBC (Java Database Connectivity) is used to connect java application with database.

It provides **classes** and **interfaces** to connect or communicate Java application with database.

JDBC is an API used to communicate **Java application** to **database** in database independent and platform independent manner.



Example
Oracle
MS Access
My SQL
SQL Server
Etc.



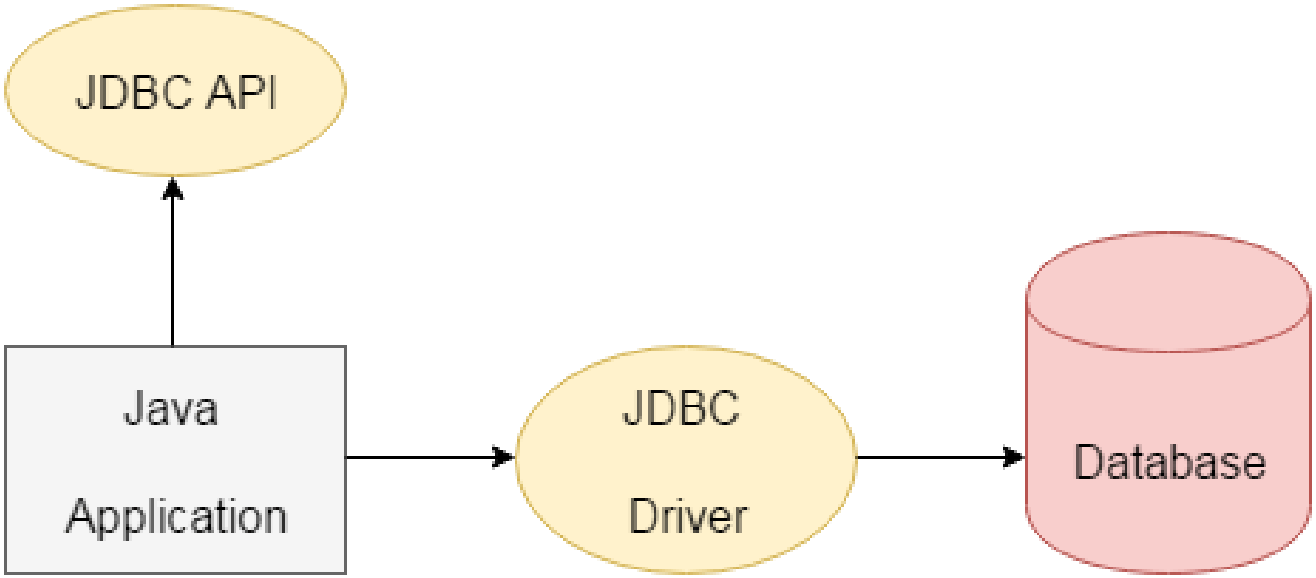
What is an API ?

- **Application Programming Interface**
- A set of routines, protocols, and tools for building software applications.
- JDBC is an API, which is used in java programming for interacting with database.

Java Database Connectivity (JDBC)

Java JDBC is a Java API to connect and execute query with the database. JDBC (Java Database Connectivity) API uses JDBC drivers to connect with the database.

We can use JDBC API to access tabular data stored into any relational database.



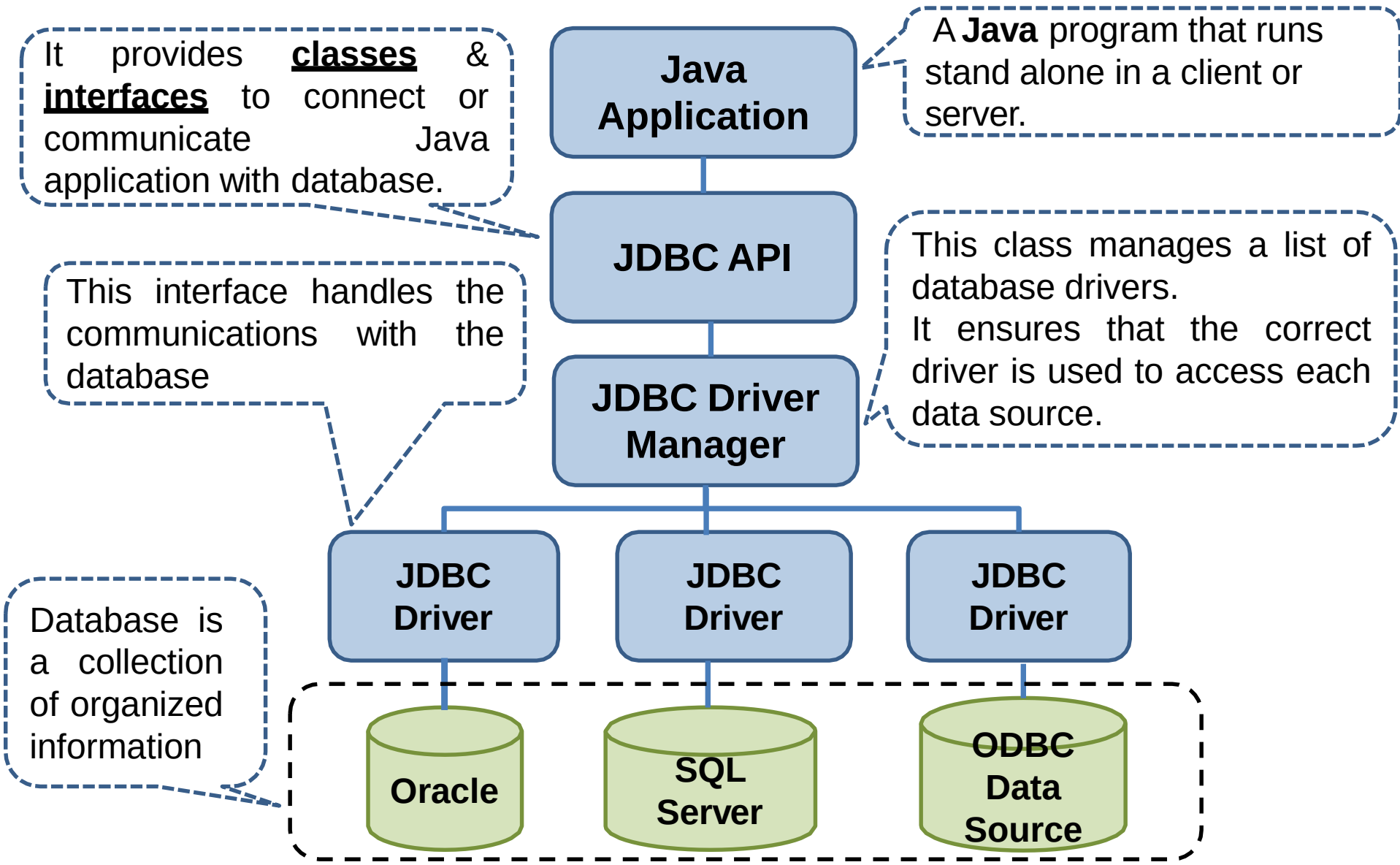
Why JDBC

Before JDBC, ODBC API was the database API to connect and execute query with the database. But, **ODBC (Open Database Connectivity)** API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

JDBC Architecture



JDBC Drivers

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

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

1) 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.

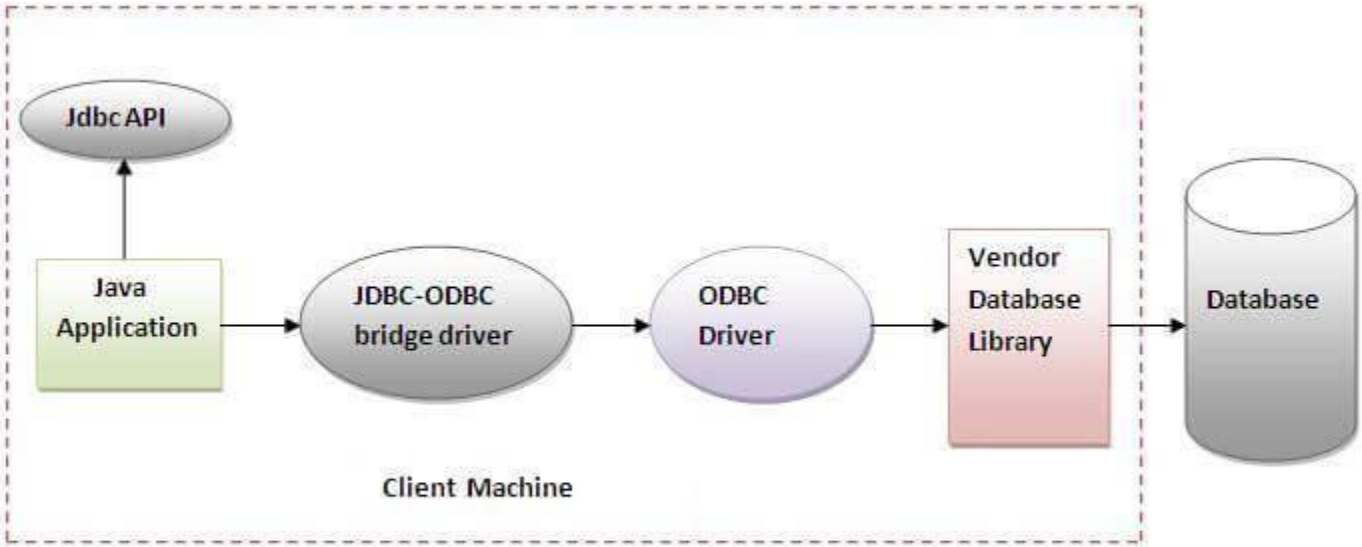


Figure- JDBC-ODBC Bridge Driver

In Java 8, the JDBC-ODBC Bridge has been removed.

1) JDBC-ODBC bridge driver

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 Bridge.

Advantages:

- ✓easy to use.
- ✓can be easily connected to any database.

Disadvantages:

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

2) Native-API 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.

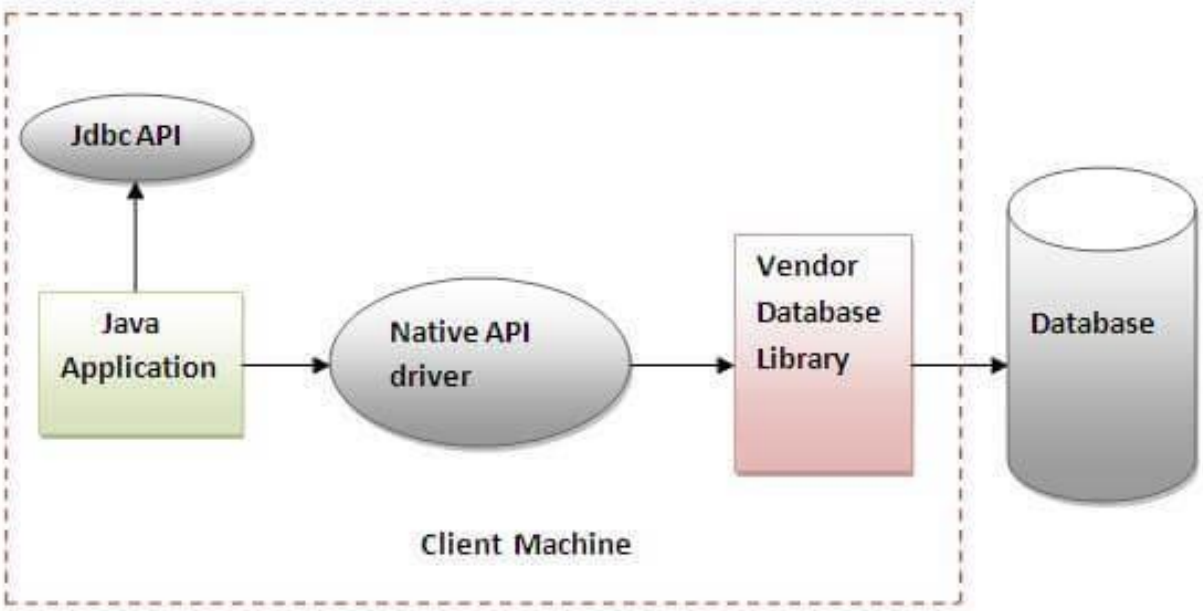


Figure- Native API Driver

2) Native-API Driver

Advantage:

✓ Performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

✓The Native driver needs to be installed on the each client machine.

✓ The Vendor client library needs to be installed on client machine.

3) Network Protocol 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.**

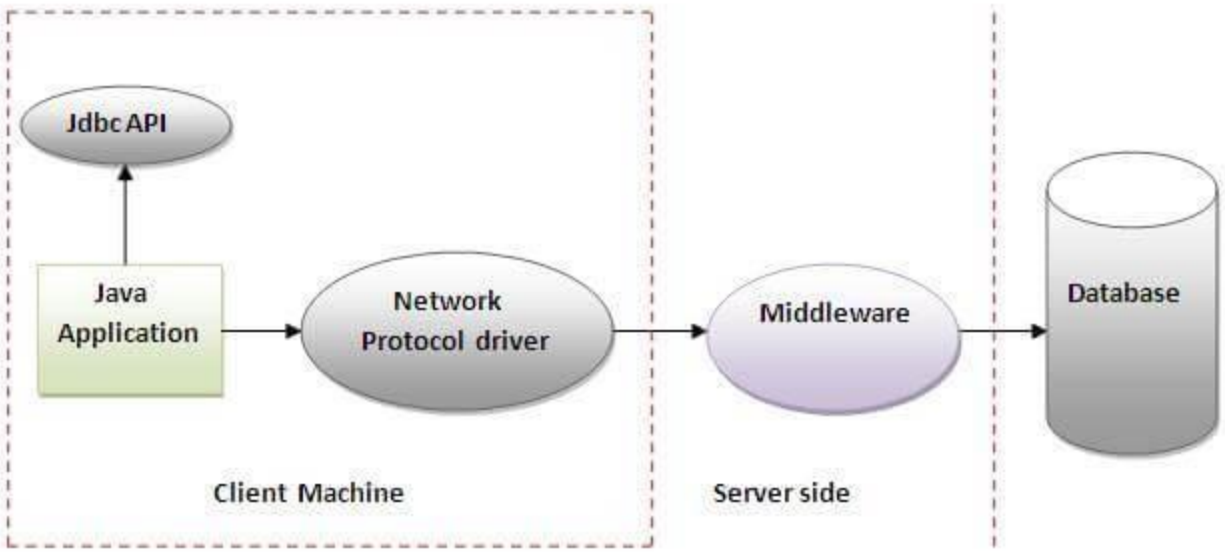


Figure- Network Protocol Driver

3) Network Protocol driver

Advantage:

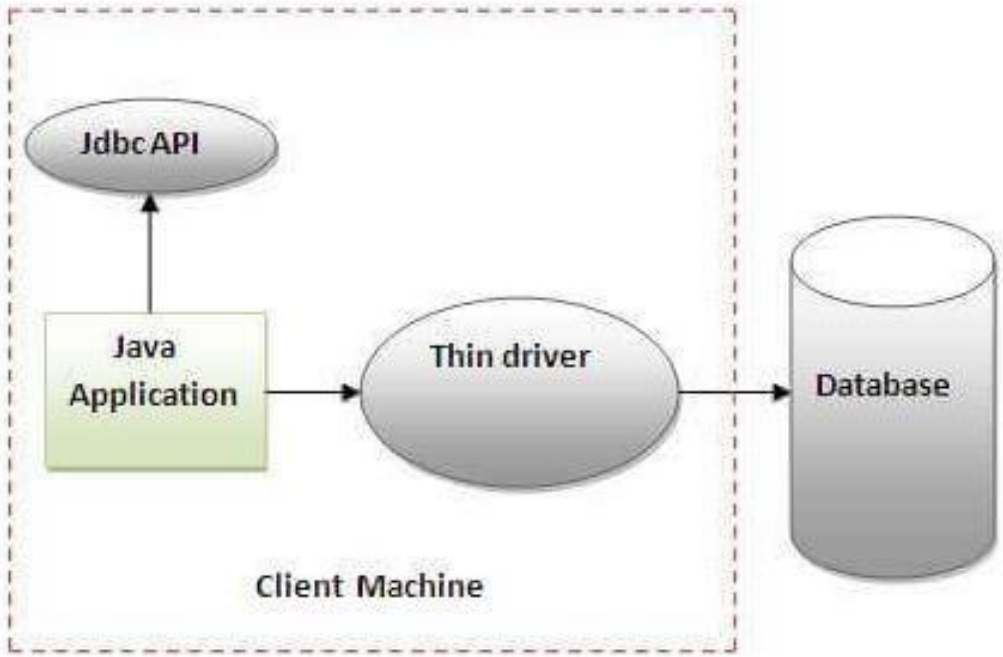
- ✓ No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

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

4) Thin 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.



4) Thin driver

Advantage:

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

Disadvantages:

- Drivers depend on the Database.

JDBC with Different RDBMS

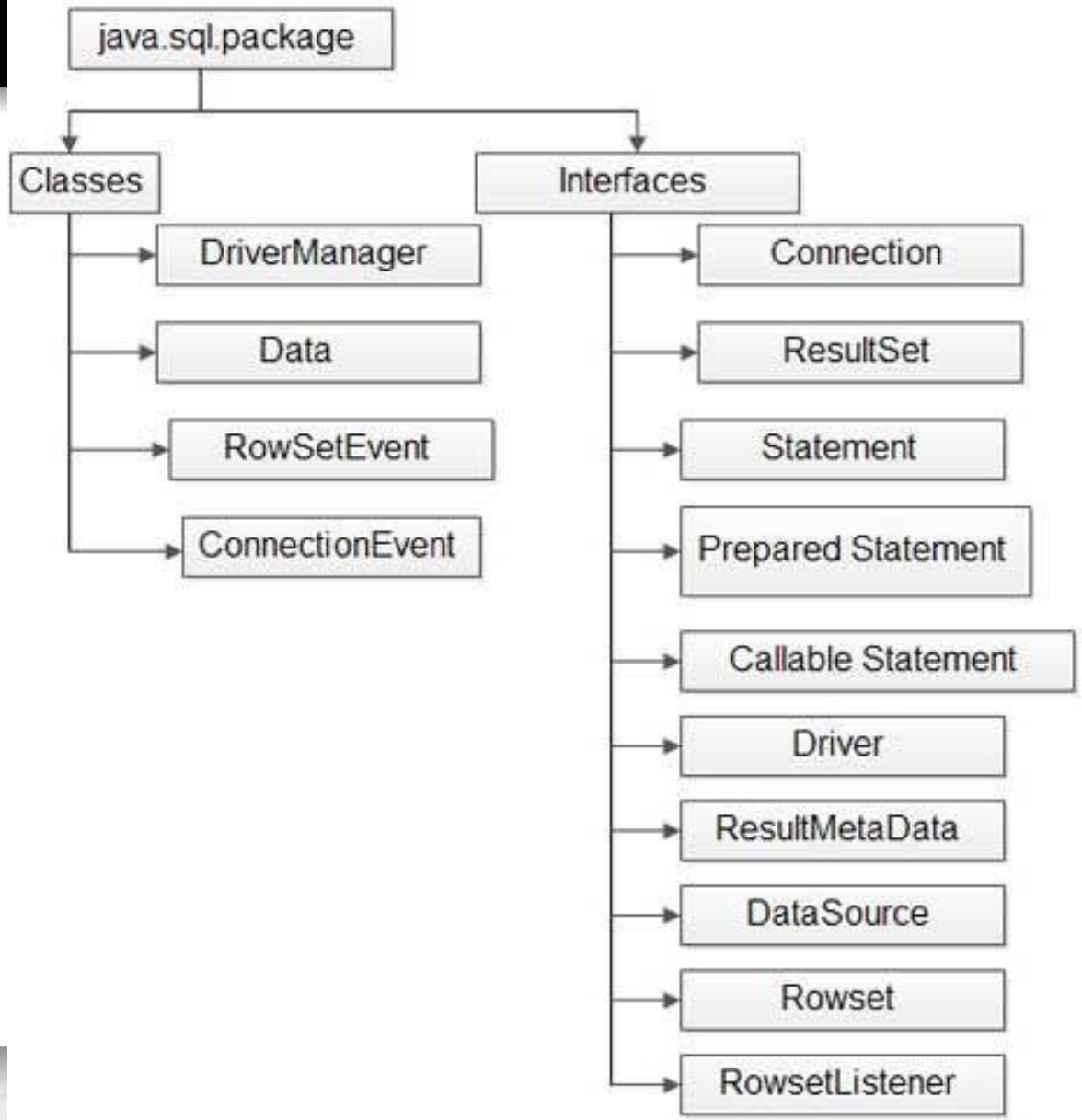
RDBMS	JDBC Driver Name	URL Format
MySQL	com.mysql.jdbc.Driver	**jdbc:mysql://**hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@**hostname:port Number:databaseName
DB2	com.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2:**hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds:**hostname: port Number/databaseName
SQLite	org.sqlite.JDBC	**jdbc:sqlite:**C:/sqlite/db/databaseName
SQLServer	com.Microsoft.sqlserver.jdbc.SQLServerDriver	**jdbc:Microsoft:sqlserver:**//hostname:1433;DatabaseName

Comparison between JDBC Drivers

Type:	Type 1	Type 2	Type 3	Type 4
Name:	**JDBC-ODBC Bridge**	**Native Code Driver/ JNI**	**Java Protocol/ Middleware**	**Database Protocol**
Example	MS Access	Oracle OCI driver	IDA Server	MySQL
Execution Speed	Slowest among all	Faster Compared to Type1	Slower Compared to Type2	Fastest among all
Driver	Thick Driver	Thick Driver	Thin Driver	Thin Driver

Major Classes and Interfaces in JDBC

Classes and Interfaces in sql package



Connectivity with 5 Steps

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

1. Register the driver class
2. Creating connection
3. Creating statement
4. Executing queries
5. Closing connection

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

Here, Java program is loading oracle driver to esteblish database connection.

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

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:mysql://hostname/ databaseName", "root","password");
```

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

Example to create the statement object

```
Statement stmt=con.createStatement();
```

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)   +   "   "   +
rs.getString("dept_name"));
}
```

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();
```

JDBC MySQL Example

Create database : MU

Table : emp (emp_id number, name varchar(30), dept varchar2(20));

Insert records

Display records from table

Creating simple JDBC Application

Demo

Netbeans with MySQL database

Refer JDBC_demo

Activity

https://www.menti.com/almkcrdxsxmx

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().

Useful methods of DriverManager class

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

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.

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.

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.

Commonly used methods of Statement interface:

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.

Batch Processing in JDBC

Methods of Statement interface

The required methods for batch processing are given below:

Method	Description
void addBatch(String query)	It adds query into batch.
int[] executeBatch()	It executes the batch of queries.

Batch Processing follows following steps:

Load the driver class
Create Connection
Create Statement
Add query in the batch
Execute Batch
Close Connection

Check Program → executeBatch_demo

ResultSet Interface

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

But we can make **this object to move forward** and **backward** direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

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

Types of ResultSet

The possible RSType are given below. If you do not specify any ResultSet type, you will automatically get one that is **TYPE_FORWARD_ONLY.**

Type	Description
ResultSet.TYPE_FORWARD_ONLY	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_SENSITIVE.	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

ResultSet Interface Methods

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.

Concurrency of Result Set

The concurrency of the ResultSet object determines whether its contents can be updated or not.

If you do not specify any Concurrency type, you will automatically get one that is CONCUR_READ_ONLY.

Concurrency	Description
ResultSet.CONCUR_READ_ONLY	Creates a read-only result set. This is the default
ResultSet.CONCUR_UPDATABLE	Creates an updateable result set.

JDBC MySQL Example : executeUpdate( )

In Previous code change the SQL query in the executeQuery

- In Previous code change the SQL query in the executeQuery

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

```
int i = stmt. executeUpdate(“create TABLE ss_table (Roll no int, Name Varchar(20))");
        if(i==0)
            System.out.println("Table created");
        else
            System.out.println("Table Not created");
```

Statement Example : Execute( ), executeUpdate( ), executeQuery( )

Marwadi education foundation

Check Program → statement_demo

Check Program → executeBatch_demo

Activity

https://www.menti.com/alj3tkji414s

Types of Statements

There are three types of statements in JDBC namely:

1. Statement
2. Prepared Statement
3. Callable statement.

Let's explore all of these in next slide…

1. Statement

Statement

The Statement interface represents the static SQL statement. It helps you to create a general purpose SQL statements using Java.

Creating a statement

You can create an object of this interface using the **createStatement()** method of the **Connection** interface.

Create a statement by invoking the **createStatement()** method as shown below.

```
stmt = conn.createStatement();
String sql1 = "update demo set Enroll=999 where Enroll=1213";
int rows = stmt.executeUpdate(sql1);
```

1. Statement

Executing the Statement object

Once you have created the statement object you can execute it using one of the execute methods namely, execute(), executeUpdate() and, executeQuery().

execute(): This method is used to **execute SQL DDL statements**, it returns a boolean value specifying whether the ResultSet object can be retrieved.

executeUpdate(): This method is used to **execute statements such as insert, update, delete.** It returns an integer value representing the number of rows affected.

executeQuery(): This method is used to **execute** statements that returns **tabular data** (example SELECT statement). It returns an object of the class ResultSet.

Refer Example: Statement_Demo

2. Prepared Statement

The **PreparedStatement** interface extends the Statement interface. It represents a precompiled SQL statement which can be executed multiple times. This accepts parameterized SQL queries and you can pass 0 or more parameters to this query.

Initially, this statement uses place holders "**?**" instead of parameters, later on, you can pass arguments to these dynamically using the **setXXX()** methods of the **PreparedStatement** interface.

2. Prepared Statement

Creating a PreparedStatement

You can create an object of the **PreparedStatement** (interface) using the **prepareStatement()** method of the Connection interface. This method accepts a query (parameterized) and returns a PreparedStatement object.

When you invoke this method the Connection object sends the given query to the database to compile and save it. If the query got compiled successfully then only it returns the object.

To compile a query, the database doesn't require any values so, you can use (zero or more) **placeholders** (Question marks "?") in the place of values in the query.

Check example in next slide…

2. Prepared Statement

For example, if you have a table named **Employee** in the database created using the following query:

```
CREATE TABLE Employee(Name VARCHAR(255), Salary INT NOT NULL, Location VARCHAR(255));
```

Then, you can use a **PreparedStatement** to insert values into it as shown below.

```
//Creating a Prepared Statement

String query="INSERT INTO Employee(Name, Salary, Location)VALUES(?, ?, ?)";

Statement pstmt = con.prepareStatement(query);
```

2. Prepared Statement

Setting values to the place holders

The **PreparedStatement** interface provides several setter methods such as setInt(), setFloat(), setArray(), setDate(), setDouble() etc.. to set values to the place holders of the prepared statement.

These methods accepts two arguments one is an integer value representing the placement index of the place holder and the other is an int or, String or, float etc… representing the value you need to insert at that particular position.

Once you have created a prepared statement object (with place holders) you can set values to the place holders of the prepared statement using the setter methods as shown below:

```
pstmt.setString(1, "Amit");
pstmt.setInt(2, 3000);
pstmt.setString(3, "Hyderabad");
```

2. Prepared Statement

Executing the Prepared Statement

Once you have created the **PreparedStatement** object you can execute it using one of the **execute()** methods of the **PreparedStatement** interface namely, execute(), executeUpdate() and, executeQuery().

execute(): This method executes normal static SQL statements in the current prepared statement object and returns a boolean value.

executeQuery(): This method executes the current prepared statement and returns a **ResultSet** object.

executeUpdate(): This method executes SQL DML statements such as insert update or delete in the current Prepared statement. It returns an integer value representing the number of rows affected.

Refer Example: Prepare_Statement

3. Callable Statement

The **CallableStatement** interface provides methods to execute stored procedures. Since the JDBC API provides a stored procedure SQL escape syntax, you can call stored procedures of all RDBMS in a single standard way.

Creating a CallableStatement
You can create an object of the **CallableStatement** (interface) using the **prepareCall()** method of the **Connection** interface.
This method accepts a string variable representing a query to call the stored procedure and returns a **CallableStatement** object.

A CallableStatement can have input parameters or, output parameters or, both. To pass input parameters to the procedure call you can use place holder and set values to these using the setter methods (setInt(), setString(), setFloat()) provided by the CallableStatement interface.

Suppose, you have a procedure name myProcedure in the database you can prepare a callable statement as: //Preparing a CallableStatement CallableStatement
cstmt = con.prepareCall("{call myProcedure(?, ?, ?)}");

3. Callable Statement

Setting values to the input parameters

You can set values to the input parameters of the procedure call using the setter methods.

These accept two arguments one is an integer value representing the placement index of the input parameter and the other is an int or, String or, float etc… representing the value you need to pass an input parameter to the procedure.

Note: Instead of index you can also pass the name of the parameter in String format.

```
cstmt.setString(1, "Raghav");
cstmt.setInt(2, 3000);
cstmt.setString(3, "Hyderabad");
```

3. Callable Statement

Executing the Callable Statement

Once you have created the CallableStatement object you can execute it using one of the **execute()** method.

```
cstmt.execute();
```

Refer Example: Callable_Statement_IN

Instance of Callable Statement

The **prepareCall()** method of Connection interface returns the instance of CallableStatement. Syntax is given below:

public CallableStatement prepareCall("{ call procedurename(?,?...?)}");

The example to get the instance of CallableStatement is given below:

CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");

It calls the procedure myprocedure that receives 2 arguments.

Parameters used with Callable Statement

Three types of parameters exist: IN, OUT, and INOUT. The PreparedStatement object only uses the IN parameter. The CallableStatement object can use all the three.
Here are the definitions of each –

Parameter	Description
IN	A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. You retrieve values from theOUT parameters with the getXXX() methods.
INOUT	A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

Difference between stored Procedure and function

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

PL/SQL Function and Procedure

Just as a Connection object creates the Statement and PreparedStatement objects, it also creates the CallableStatement object, which would be used to execute a call to a database stored procedure.

Create Procedure in MySQL:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `MU`.`getStudentName` $$
CREATE PROCEDURE `MU`.`getStudentName`
(IN VAR_SID INT, OUT VAR_SNAME VARCHAR(20))
BEGIN
        SELECT sname INTO VAR_SNAME FROM student
        WHERE SID= VAR_SID;
 END $$
DELIMITER ;
```

Check Programs → Callable_Statement

Transaction Management in JDBC

Transaction represents a single unit of work.

The **ACID properties** describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.

Atomicity means either all successful or none.

Consistency ensures bringing the database from one consistent state to another consistent state.

Isolation ensures that transaction is isolated from other transaction.

Durability means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Transaction Management in JDBC

In JDBC, **Connection interface** provides methods to manage transaction.

Method	Description
void setAutoCommit(boolean status)	It is true bydefault means each transaction is committed bydefault.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

Check Program → TransactionDemo

JDBC Rowset Interface

The instance of **RowSet** is the java bean component because it has properties and java bean notification mechanism. It is introduced since JDK 5.

It is the **wrapper** of ResultSet. It holds tabular data like ResultSet but it is easy and flexible to use.

The implementation classes of RowSet interface are as follows:

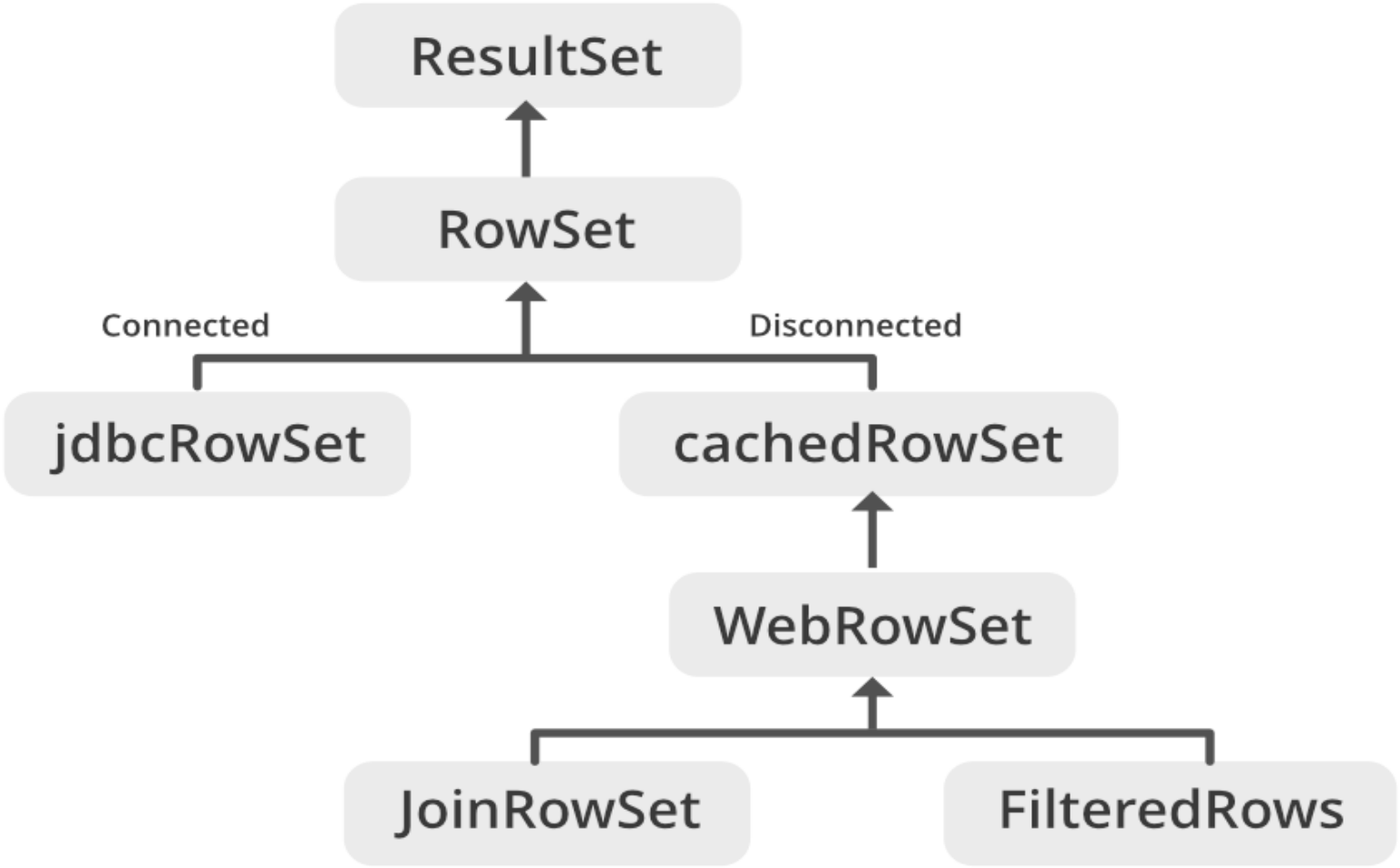
1. JdbcRowSet
2. CachedRowSet
3. WebRowSet
4. JoinRowSet
5. FilteredRowSet

Check Programs → RowsetExample, RowsetExample1

Advantage of RowSet

- It is easy and flexible to use
- It is Scrollable and Updatable by default

JDBC Rowset Interface



JDBC Rowset Interface


```
import javax.sql.rowset.JdbcRowSet;     import javax.sql.rowset.RowSetProvider;

public class RowSetExample {
      public static void main(String[] args) throws Exception {
            Class.forName("com.mysql.jdbc.Driver");

   //Creating and Executing RowSet
      JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
      rowSet.setUrl("jdbc:mysql://localhost/MU");
      rowSet.setUsername("root");
      rowSet.setPassword("");

      rowSet.setCommand("select * from student");
      rowSet.execute();

   while (rowSet.next()) {  // Generating cursor Moved event
                  System.out.println("Student Id: " + rowSet.getString(1));
            }  }  }
```

JDBC RowSet vs ResultSet

ResultSet	RowSet
A ResultSet always maintains connection with the database.	A RowSet can be connected, disconnected from the database.
It cannot be serialized.	A RowSet object can be serialized.
ResultSet object cannot be passed other over network.	You can pass a RowSet object over the network.
ResultSet object is not a JavaBean object You can create/get a result set using the **executeQuery()** method.	ResultSet Object is a JavaBean object. You can get a RowSet using the **RowSetProvider.newFactory().createJdb cRowSet()** method.
By default, ResultSet object is not scrollable or, updatable.	By default, RowSet object is scrollable and updatable.

Mini Project 1

- Using JDBC connection, Create CRUD Application and perform Insert, select, update and delete operations.
- MYSQL table should have minimum four columns and 5 records.

Summary

- JDBC Architecture
- Types of JDBC Drivers
- Introduction to major JDBC Classes and Interface
- Creating simple JDBC Application
- Types of Statement
- Exploring ResultSet Operations
- Batch Updates in JDBC
- Creating CRUD Application
- Using Rowsets Objects
- Managing Database Transaction

Up Next

- Swing features
- Swing Containers
  - Jframe
  - Jpanel
  - JWindow
- Swing components : JLabel, ImageIcon, JTextField, JButton, JToggeleButton, JCheckBox, JRadioButton, JTabbedPane, JScrollPane, JList, JComboBox, JTree, JTable
- Layout managers
- Event model in Java
- Event classes
- Event listeners
- Adapter classes

END OF UNIT-1