Java Technologies Week 5

Prepared for: DDU CE Semester 4

Topics

- Why Database connectivity is required?
- Introduction to JDBC
- What JDBC Do?
- JDBC Architecture
- JDBC driver types
- JDBC API
- Steps to create a JDBC application
- JDBC Data Types
- Statement Interface
- Preparedstatement Interface
- CallableStatement Interface
- Resultset interface
- Batch Processing

Why Database connectivity is required?

```
Class student

{

      Int id;

      String name;

      Int percentage;

}
```

The data stored in program is lost once the execution is completed.

```
Class Demo

{Public static void mai

{     Student obj=new student();

      System.out.println(obj.id);

      System.out.println(obj.name);

      System.out.println(obj.per);

}

}
```

The database connectivity is required for permanent storage of the data , so the data can be used in future.

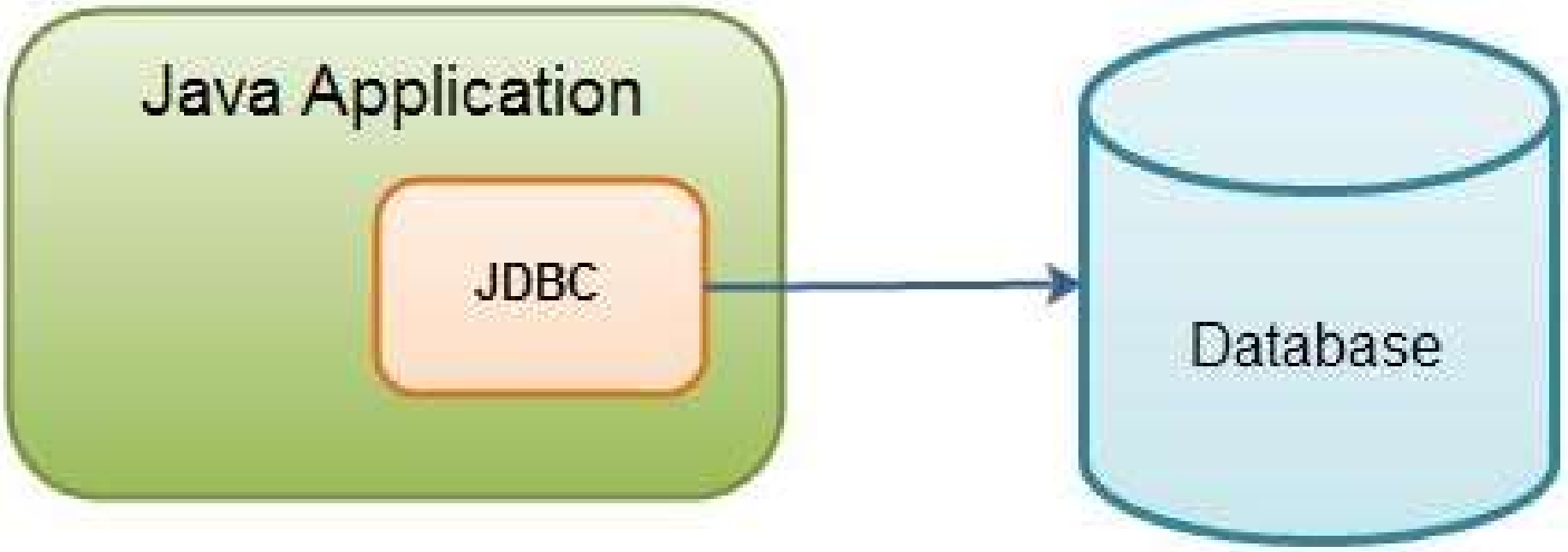
Introduction to JDBC

- JDBC stands for Java Database Connectivity, it is an API which is used in Java programming to interact with databases.
- JDBC can work with any database as long as proper drivers are provided.

- A JDBC driver is a JDBC API implementation used for connecting to a particular type of database.

- To connect to a database, we simply have to initialize the driver and open a database connection.

JDBC(Java Database Connectivity)



Java Database Connectivity

- Java Application cannot directly communicate with a database to submit data & retrieve the results of queries. This is because a database can interpret only SQL statements & not java language statements. So JDBC is a mechanism to translate java statements into SQL statements.
- JDBC is a Java API for executing SQL statements.

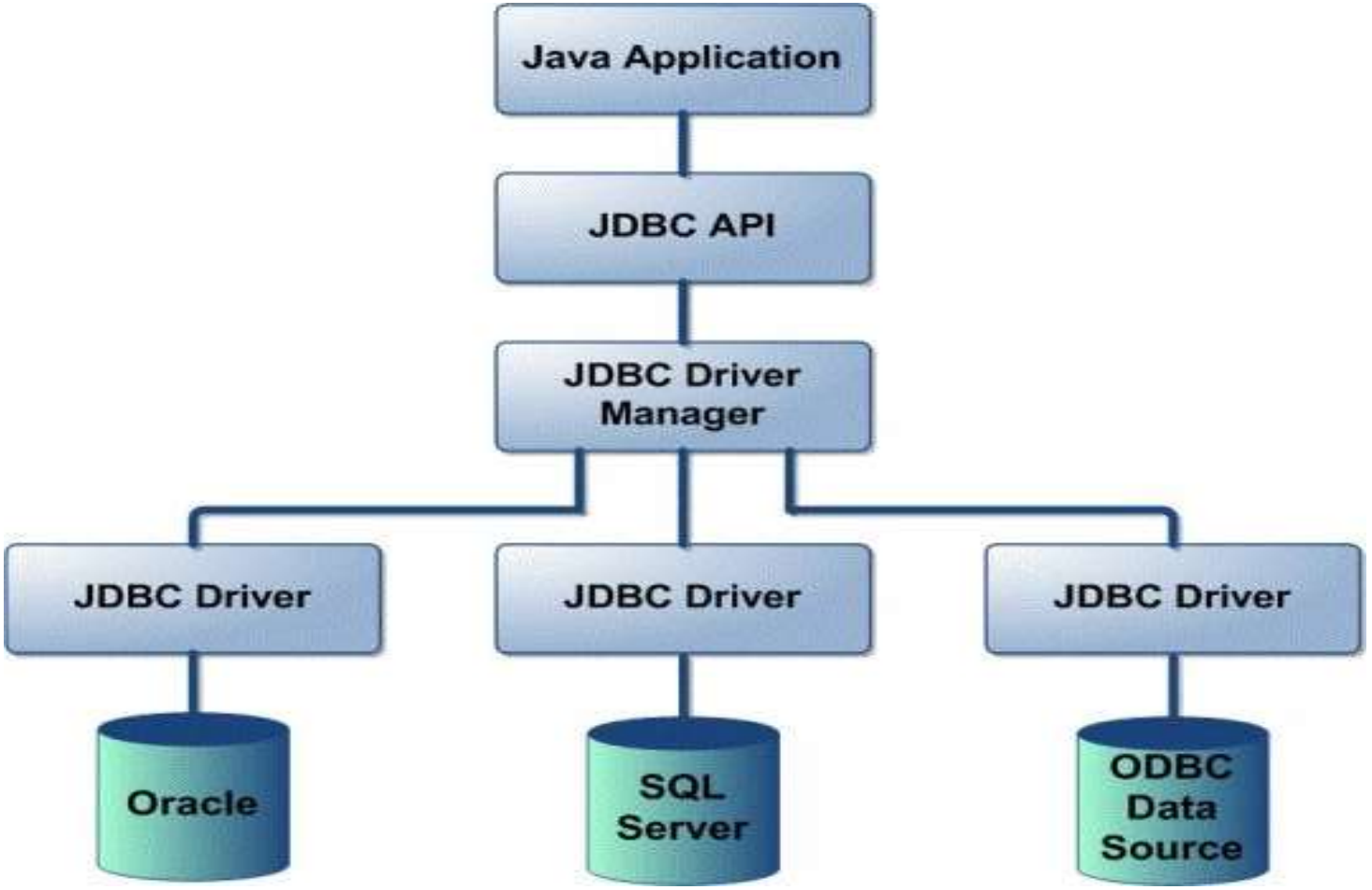
- JDBC is an application programming interface (API) for the programming language **Java**, that defines how a client may access a **database**.

- It is part of the **Java** Standard Edition platform, from Oracle Corporation.
- Sun Microsystems released JDBC as part of JDK 1.1 on February 19, 1997

Java Database Connectivity

- What JDBC Do?
  - establish a connection with database
  - Send SQL statements
  - Process the results on database side
  - Retrieve the output to java application

JDBC Architecture



JDBC Architecture

Components of JDBC

- **The JDBC API** – it provides various methods and interfaces for easy and effective communication with the databases.

- **The JDBC DriverManager** – it is a backbone of JDBC architecture. it connects a Java application to correct JDBC driver.

- **The JDBC test suite** – it evaluates the JDBC driver for its compatibility with java EE.

- **The JDBC-ODBC bridge** – it connects database drivers to the database. This bridge translate JDBC method calls to ODBC function calls.

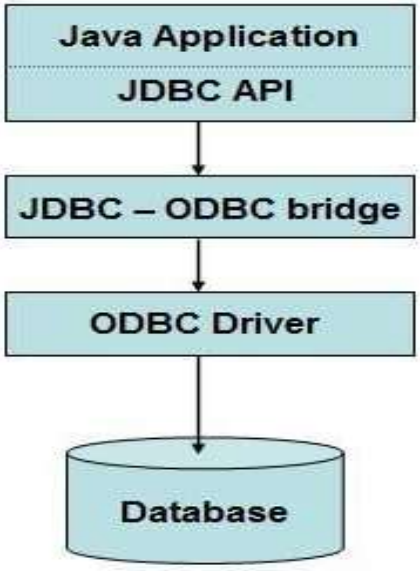
JDBC Driver types

There are four kind of JDBC drivers available.

1. Type 1 Driver – JDBC – ODBC bridge
2. Type 2 Driver – Native API Partly Java Driver
3. Type 3 Driver – Pure Java Driver for Database Middleware
4. Type 4 Driver – Pure Java Driver

Type-1 Driver

- The Type – 1 driver acts as a bridge between JDBC and other database connectivity mechanisms, such as ODBC.
- An example of this type of driver is the Sun JDBC-ODBC driver



Type 1: JDBC-ODBC Bridge

Type-1 Driver

- JAVA App makes call to JDBC ODBC Bridge Driver
- JDBC ODBC Bridge Driver resolves JDBC Call and makes equivalent ODBC call to ODBC driver.
- ODBC Driver Completes the request and sends response to JDBC ODBC Bridge Driver.
- JDBC ODBC Bridge Driver converts the response to JDBC Standards and Displays the result into requesting JAVA App.
- Ex: Sun JDBC ODBC Bridge Driver

Type-1 Driver

Advantages:

- Represent single driver implementation to interact with different data source
- Allow you to communicate with all the databases supported by the ODBC driver

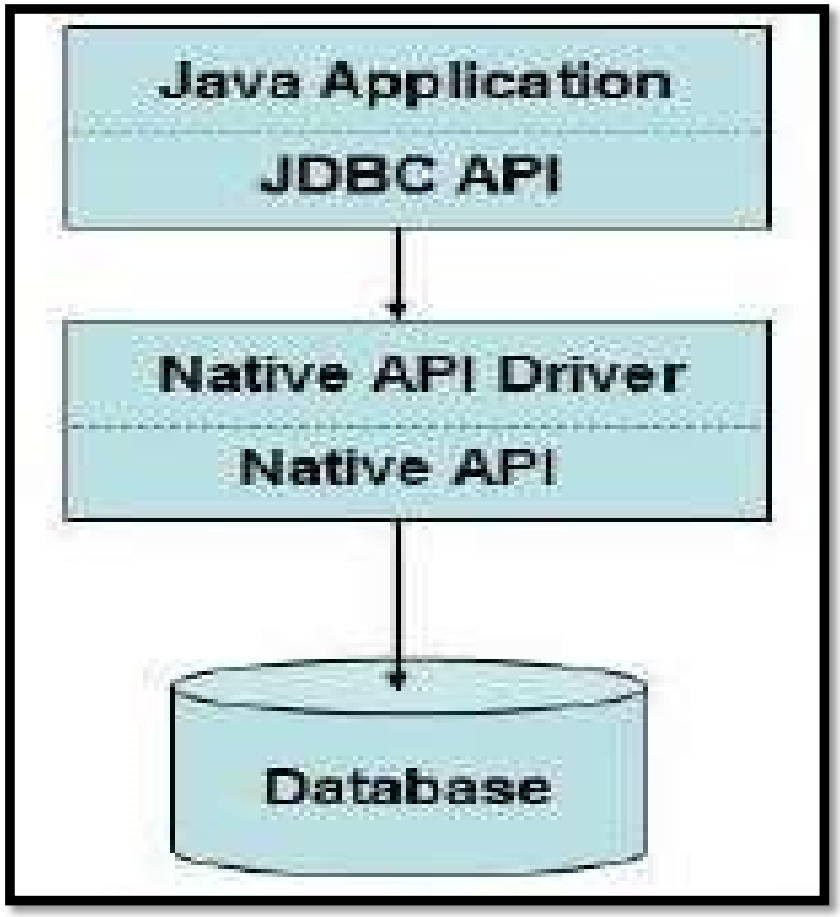
Disadvantages:

- Decreases the execution speed due to a large number of translations.
- It depends on ODBC driver.
- Requires the ODBC binary code or ODBC client library that must be installed on every client.

TYPE -2 DRIVER
Native API Partly Java Driver

- Follows 2- Tier Architecture
- JDBC calls are converted into database specific native calls in the Client Machine & Request Dispatched to database Specific native Libraries.
- Native Database Libraries sends requests to database server by using native protocol

TYPE -2 DRIVER



TYPE -2 DRIVER

Advantages :

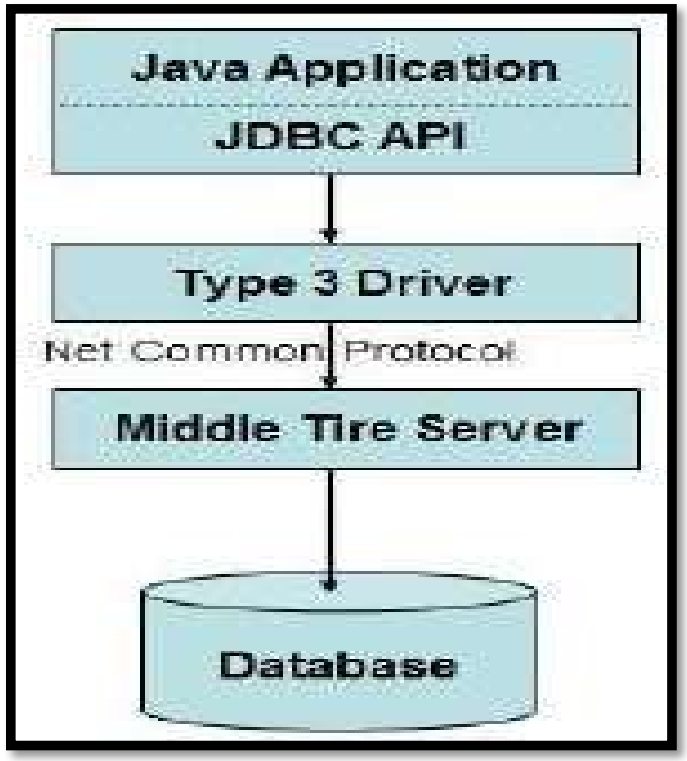
- Helps to access the data faster as compared to other types of drivers.

Disadvantages :

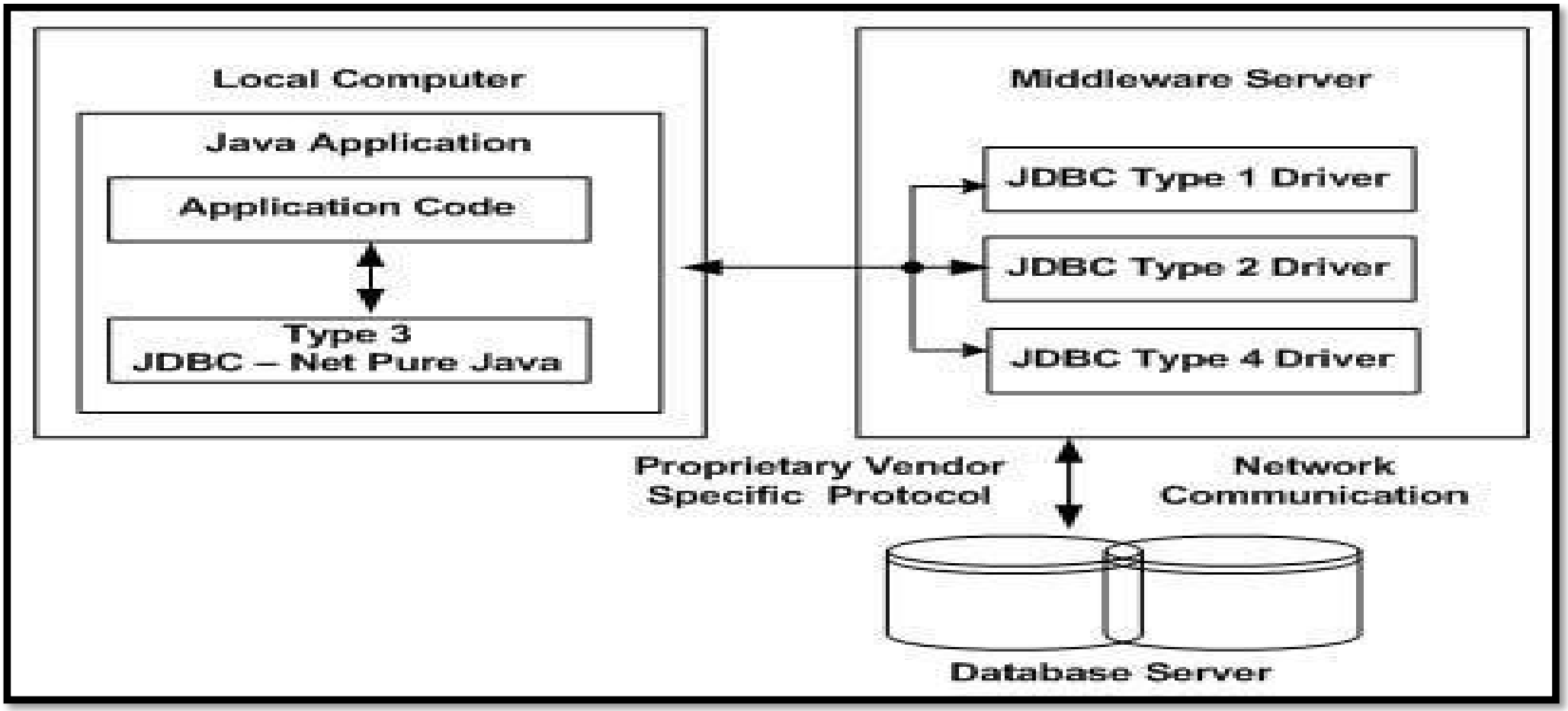
- Requires native libraries to be installed on client machines.
- Any bug in the Type-2 driver might crash the JVM.
- Increase the cost of the application in case it is run on different platforms.

TYPE-3 DRIVER
Pure Java Driver for Database Middleware

- Translates JDBC call to Direct server calls with the help of middleware server.
- Also known as Net Protocol Fully JAVA Driver



TYPE-3 DRIVER



TYPE-3 DRIVER

Advantages :

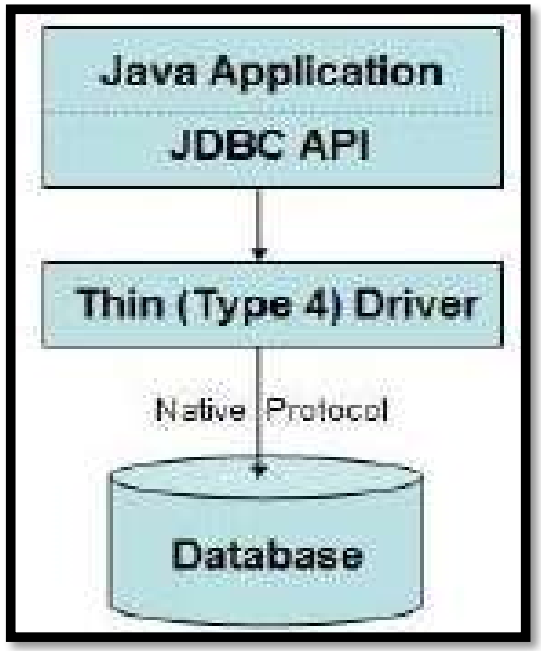
- Serves as all Java driver.
- Does not require any native library to be installed on the client machine.
- Ensures database independency.
- Does not provides database location information.
- Switch over from one database to another without changing the code.

Disadvantages:

- Perform the task slowly.
- Costlier.

TYPE- 4 DRIVER
Pure Java Driver

- Implements Database Protocol to Directly deal with the data base
- Do not require native libraries
- Translates JDBC class to direct database specific calls
- Also known as THIN Drivers.



TYPE- 4 DRIVER

Advantages:

- Pure Java driver.
- Does not require any native library to be installed on the client machine.
- Uses database server specific protocol.
- Does not require middleware server.

Disadvantages:

- It is DBMS vendor dependent

Which Driver should be used?

- If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.
- If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
- Type 2 drivers are useful in situations where a type 3 or type 4 driver is not available yet for your database.
- The type 1 driver is not considered a deployment-level driver and is typically used for development and testing purposes only.

Comparison of JDBC Drivers

	Type 1	Type 2	Type 3	Type 4
1	not portable	not portable	portable	portable
2	platform dependent	platform dependent	platform independent	platform independent
3	database independent	database dependent	database independent	database dependent
4	not completely written in java	not completely written in java	completely written in java	completely written in java
5	slowest of all drivers	faster than Type 1	most efficient amongst all driver types	quite efficient
6	The client system requires the ODBC Installation	Database specific Native API must be installed in the Client System	Server-based driver, so there is no need for any vendor database library on client machines.	No need to install special software on the client or server, different driver for each database can be downloaded automatically
7	not suitablefor web applications	not suitable for web applications	suitable for the web	most suitable for the web.

JDBC APIs

The java.sql Package

- It also known as the JDBC core API.
- This package includes the interfaces and methods to perform JDBC core operations, such as creating and executing SQL queries.
- The java.sql package consists of the interfaces and classes that need to be implemented in an application to access a database.
  - Connection Management
  - Database access
  - Data Types
  - Exception and Warning

The java.sql Package (cont…)

- Connection Management classes
  - Java.sql.Connection
  - Java.sql.Driver
  - Java.sql.DriverManager
  - Java.sql.DriverPropertyInfo
  - Java.sql.SQLPermission
- Database Access Interfaces
  - Java.sql.CallableStatement
  - Java.sql.PreparedStatement
  - Java.sql.ResultSet
  - Java.sql.Statement and java.sql.Savepoint

Steps to create a JDBC application

1. Register the driver
2. Create connection (to database)
3. Create statement (to execute queries)
4. Execute queries
5. Processing data returned by the DBMS
6. Close the connection

3.Executing SQL Statements

- A **Statement** is an interface that represents a SQL statement.
- Executing the Statement objects, **ResultSet** objects are generate, which is a table of data representing a database result set.
- Types of statements:
  1. **Statement**: Used to implement simple SQL statements with no parameters.
  2. **PreparedStatement**: (Extends Statement.) Used for precompiling SQL statements that might contain input parameters.
  3. **CallableStatement**: (Extends PreparedStatement.) Used to execute stored procedures that may contain both input and output parameters.

4.Executing Queries

- **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.

- **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.

JDBC data Types

- The JDBC driver converts the Java data type to the appropriate JDBC type, before sending it to the database. It uses a default mapping for most data types.
- For example, **a Java int is converted to an SQL INTEGER**. Default mappings were created to provide consistency between drivers.
- The following table summarizes the default JDBC data type that the Java data type is converted to, when you call the setXXX() method.
- ResultSet object provides corresponding getXXX() method for each data type to retrieve column value. Each method can be used with column name or by its ordinal position.

JDBC data Types

SQL	JDBC/Java	setXXX	getXXX
VARCHAR	java.lang.String	setString	getString
CHAR	java.lang.String	setString	getString
LONGVARCHAR	java.lang.String	setString	getString
BIT	Boolean	setBoolean	getBoolean
NUMERIC	java.math.BigDecimal	setBigDecimal	getBigDecimal
INTEGER	Int	setInt	getInt
FLOAT	Float	setFloat	getFloat
DOUBLE	Double	setDouble	getDouble
DATE	java.sql.Date	setDate	getDate
TIME	java.sql.Date	setDate	getDate
CLOB	java.sql.Clob	setClob	getClob
BLOB	java.sql.Blob	setBlob	getBlob

Statement Interface

- Execution of Statement object

1. The executeXXX() method is invoked on the Statement object by passing the SQL statement as parameter.
2. It submit the Query to the Database.
3. The DBMS compile the given SQL statement.
4. The DBMS execute the given SQL statement.
5. If the SQL statement is SELECT statement then the database caches the results of the SQL statement in the buffer.
6. The results are sent to the Statement object.
7. Finally, the response is sent to the Java application in the form of ResultSet.

2.PreparedStatement

- Execution of PreparedStatement object

1. The prepareStatement() method of the connection object is used to get the object of the PreparedStatement interface.
2. The Connection object submits the given SQL statement to the database.
3. The database compiles the given SQL statement.
4. An execution plan is prepared by the database to execute the SQL statement.
5. The database stores the execution plan with a unique ID and returns the identity to the Connection object.

PreparedStatement

- Execution of PreparedStatement object

- In normal Statement object SQL query will always compile when it is going to executed.
- If same query will compile more than one time than it will become overhead.
- SQL query can be precompiled and executed using PreparedStatement object.
- It also has method like executeQuery() and executeUpdate() methods.

PreparedStatement

Advantages

- Improves the performance of an application as compared to the Statement object that executes the same query multiple times.
- Provide a programmatic approach to set the values.

CallableStatement

- The CallableStatement interface extends the PreparedStatement interface and also provides support for both input as well as output parameters.
- It provides a standard abstraction for all the data sources to call stored procedures and functions, irrespective of the vendor of the data source.
- This interface is used to access, invoke and retrieve the results of SQL stored procedures, functions, and cursors.

CallableStatement

- Broad-level steps to use the CallableStatement

1. Create the CallableStatement Object
2. Setting the values of the parameters
3. Registering the OUT parameters type
4. Executing the procedure of function
5. Retrieving the parameter values.

CallableStatement

- A stored procedure encapsulates the values of the following types of parameters.
- **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.

- **IN OUT** – A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

CallableStatement

- **How to create procedure**

```
Create or replace procedure insertData (rno number, name varchar2,
 address varchar2) is
  Begin
  insert into student values(rno, name, address);
  End;
```

CallableStatement

- How to write down code…
- CallableStatement cs= con.prepareCall ("{call insertData (?,?,?)}");

- **// set IN parameters**

1. cs.setInt(1, 12);
2. cs.setInt(2, “Jay”);
3. cs.setString(3, “xyz….");
4. cs.execute();

CallableStatement

- **OUT parameter**

```
Create or replace procedure getBalance (acno number, bal OUT number)
    is
Begin
select bal into amt from bank where accno=acno;
End;
```

CallableStatement

- CallableStatement (cont…)
- OUT parameter (cont…)

```
CallableStatement cs= con.prepareCall("{call
getBalance(?,?)}");
    cs.setInt(1, 101));
    cs.execute();
    System.out.println("Balance : "+ cs.getDouble(2));

con.close();
```

Resultset Interface

- **createStatement(RSType, RSConcurrency);**

- Type of ResultSet
  - ResultSet.TYPE_FORWARD_ONLY
  - ResultSet.TYPE_SCROLL_INSENSITIVE
  - ResultSet.TYPE_SCROLL_SENSITIVE.

- Concurrency of ResultSet
  - ResultSet.CONCUR_READ_ONLY
  - ResultSet.CONCUR_UPDATABLE
- Example

```
Statement stmt = conn.createStatement( ResultSet.TYPE_FORWARD_ONLY,
 ResultSet.CONCUR_READ_ONLY);
```

Batch Processing

- Batch Processing allows you to group related SQL statements into a batch and submit them with one call to the database.
- The **addBatch()** method of Statement, PreparedStatement, and CallableStatement is used to add individual statements to the batch.
- The **executeBatch()** is used to start the execution of all the statements grouped together.
- The **executeBatch()** returns an array of integers, and each element of the array represents the update count for the respective update statement.
- Just as you can add statements to a batch for processing, you can remove them with the **clearBatch()** method. This method removes all the statements you added with the addBatch() method.

Example

```
Statement stmt = conn.createStatement();
String SQL = "INSERT INTO Emp VALUES(2,‘Sachin',  ‘account’)";
stmt.addBatch(SQL);

String SQL = "INSERT INTO Emp VALUES(3,'Raj', ‘hr’)";
stmt.addBatch(SQL);

String SQL = "UPDATE Emp  SET name = ‘Kumar’  WHERE emp_id = 10";
stmt.addBatch(SQL);

int[] count = stmt.executeBatch();
```

Example

```
String SQL = "INSERT INTO Emp VALUES(?, ?, ?)”;
PreparedStatemen pstmt = conn.prepareStatement(SQL);
pstmt.setInt( 1, 40 );
pstmt.setString( 2, “smith" );
pstmt.setString( 3, “finance" );
pstmt.addBatch();

pstmt.setInt( 1, 401 );
pstmt.setString( 2, “Pawan" );
pstmt.setString( 3, “admin" );
pstmt.addBatch();
int[] count = stmt.executeBatch();
```