JDBC (Java Database Connectivity)

By

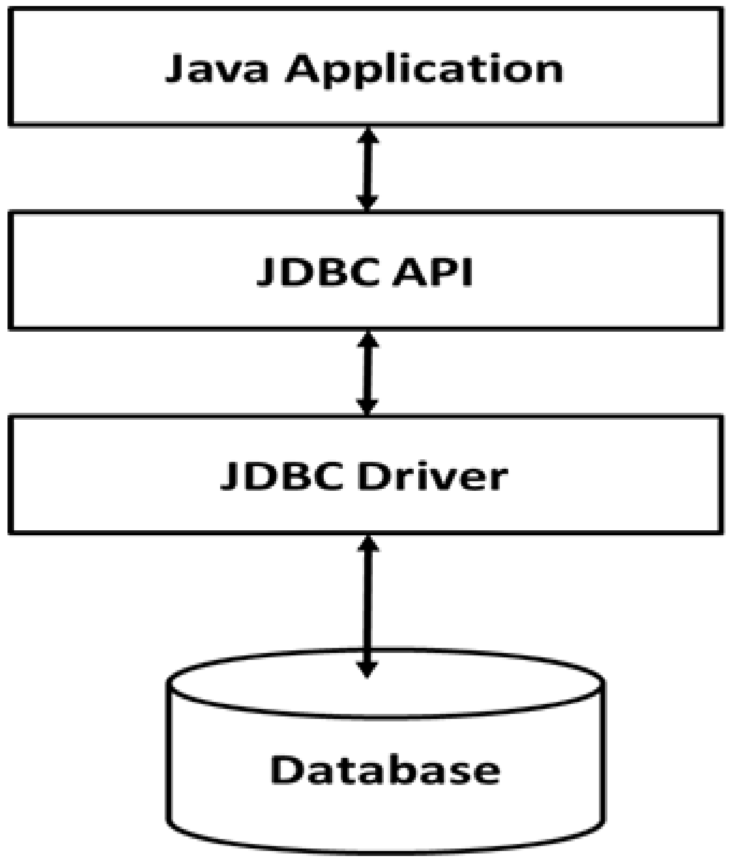
Srikanth Pragada

What is JDBC?

- ❑ The JDBC API provides universal data access from Java language.
- ❑ JDBC is Java API to execute SQL Statements.
- ❑ It is standard API to access databases from Java.
- ❑ Using JDBC API you can access any database that may run on any platform.
- ❑ The JDBC 4.x API is divided into two packages: java.sql and javax.sql.
- ❑ Both packages are included in the Java SE and Java EE platforms.

JDBC Driver

- ❑ To use the JDBC API with a particular DBMS, you need a JDBC technology-based driver to access database.
- ❑ Driver must support at least ANSI SQL-2 Entry Level (1992).



JDBC Components

JDBC provides the following components as part of the JDK.

JDBC driver manager	The JDBC driver manager is the backbone of the JDBC architecture. It actually is quite small and simple; its primary function is to connect Java applications to the correct JDBC driver and then get out of the way.
JDBC-ODBC bridge	The JDBC-ODBC bridge allows ODBC drivers to be used as JDBC drivers. It provides a way to access less popular DBMS if JDBC driver is not implemented for it.

Important Interfaces in JDBC

- ❑ The following are the interfaces provided with JDBC API.
- ❑ These interfaces are implemented by driver.
- ❑ A driver contains a collection of classes to implement these interfaces.

Interface	**Meaning**
CallableStatement	The interface used to execute SQL stored procedures.
Connection	A connection (session) with a specific database.
DatabaseMetaData	Comprehensive information about the database as a whole.
Driver	The interface that every driver class must implement.
PreparedStatement	An object that represents a precompiled SQL statement.
ResultSet	A ResultSet provides access to a table of data.
ResultSetMetaData	An object that can be used to find out about the types and properties of the columns in a ResultSet.
Statement	The object used for executing a static SQL statement and obtaining the results produced by it.

Important Classes

- ❑ The following are the classes provided by JDBC API.
- ❑ These classes are provided in addition to interfaces mentioned above.

Class	**Description**
Date	A thin wrapper around a millisecond value that allows JDBC to identify this as SQL DATE.
DriverManager	The basic service for managing a set of JDBC drivers.
DriverPropertyInfo	Driver properties for making a connection.
Time	A thin wrapper around *java.util.Date* that allows JDBC to identify this as SQL TIME value.
Timestamp	This class is a thin wrapper around *java.util.Date* that allows JDBC to identify this as SQL TIMESTAMP value.
Types	The class that defines constants that are used to identify generic SQL types, called JDBC types.

DriverManager Class

- Part of java.sql package, used to manage JDBC drivers.
- Sits between the application programs and the drivers.
- Keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.

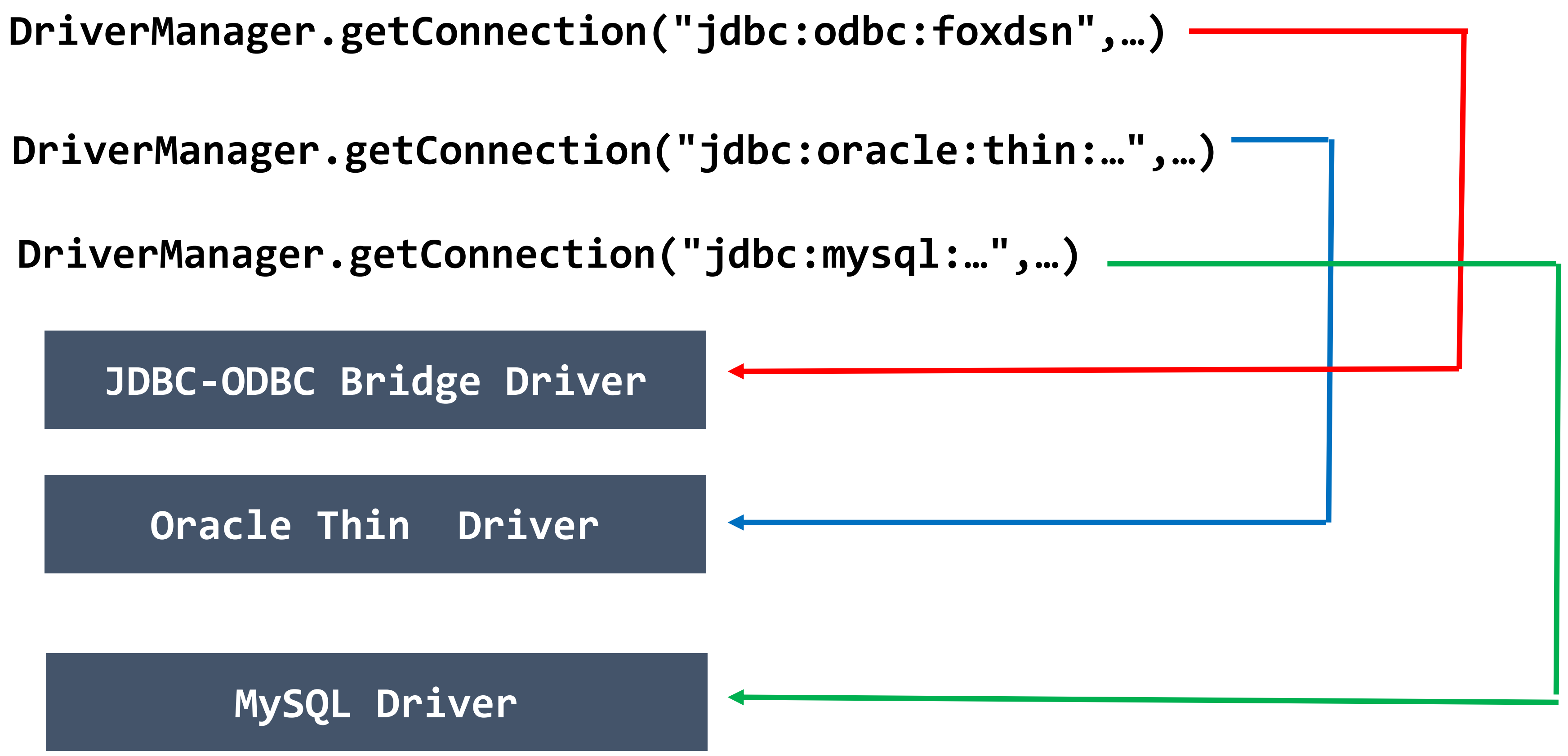
Method	Meaning
Connection getConnection (String url, String un, String pwd)	Establishes a connection with the specified database.
Driver getDriver(String url)	Returns a driver that can understand the URL.
Enumeration getDrivers()	Returns the drivers that are currently loaded.
void registerDriver (Driver driver)	Registers the given driver.

Driver Interface

- ❑ This is the interface that every driver has to implement.
- ❑ Each driver should supply a class that implements the Driver interface.

Method	Meaning
boolean acceptsURL(String url)	Returns true if the driver thinks that it can open a connection to the given URL.
Connection connect (String url, Properties info)	Connects to a database and returns connection object.
int getMajorVersion()	Returns major version number.
int getMinorVersion()	Returns minor version number.

Using JDBC Drivers



```
DriverManager.getConnection("jdbc:odbc:foxdsn",…)

DriverManager.getConnection("jdbc:oracle:thin:…",…)

DriverManager.getConnection("jdbc:mysql:…",…)
```

Connection Interface

- ❑ Represents a connection to specific database.
- ❑ SQL statements are executed and results are returned within the context of a connection.

Method	Meaning
Statement createStatement()	Creates and returns an object of Statement
Statement createStatement (int resultSetType, int resultSetConcurrency)	Creates a Statement object that will generate ResultSet objects with the given type and concurrency
DatabaseMetaData getMetaData()	Returns an object of DatabaseMetaData, which can be used to get information about the database
boolean isClosed()	Returns true if connection is closed
CallableStatement prepareCall(String sql)	Creates a callable statement
PreparedStatement prepareStatement (String sql)	Creates a prepared statement
void close()	Closes the connection

Connecting to MS SQL Server

- ❑ Make sure JDBC Driver for MS SQL Server (.jar) is available in classpath by adding it using **Libraries** tab in **Java Build Path**.
- ❑ MS SQL Server JDBC Driver is mssql-jdbc-11.2.0-jre17.jar

TestConnection.java

```
import java.sql.Connection;
import java.sql.DriverManager;

public class TestConnection {
    public static void main(String[] args) throws Exception {
        Connection con = DriverManager.getConnection
           ("jdbc:sqlserver://srikanthlaptop\\sqlexpress:1433;database=msdb;
               user=sa;password=sa;encrypt=true;trustServerCertificate=true");
        System.out.println(con.getClass());
        System.out.println("Connected To MSDB Database!");
        con.close();
    }
}
```

Statement Interface

- Is used to execute an SQL command.
- At the time of creating Statement we have to specify what type of ResultSet is to be created from this Statement.
- Only one ResultSet per statement can be opened at a time.

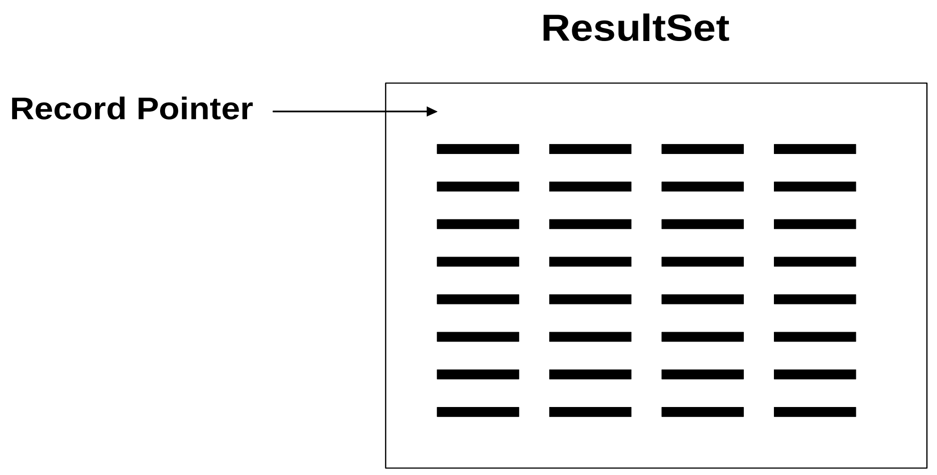
Method	Meaning
ResultSet executeQuery(String)	Executes the query in the statement and returns ResultSet.
int executeUpdate(String)	Executes DML command and returns the no. of rows updated.
boolean execute(String)	Executes SQL command and returns true if query is executed.
Connection getConnection()	Returns the connection object that produced this statement.
ResultSet getResultSet()	Returns current ResultSet.
int getUpdateCount()	Returns update count of most recently executed command.
void close()	Enables the resources to be released.

Executing SQL Command

- ❑ Establish a connection to database.
- ❑ Create a statement using **createStatement**() method of Connection interface.
- ❑ Execute an SQL statement using one of the methods of Statement interface.
- ❑ If SQL command is a query, use ResultSet to access data retrieved from database.

ResultSet Interface

- Provides access to a set of rows.
- Contains a cursor that points to a particular record in the ResultSet.
- This cursor is positioned initially before the first row.
- You can retrieve values of columns using get<type>() methods. Columns can be referred either by column number or name.
- Columns are numbered from 1.
- A ResultSet may be either scrollable or forward only. It may be either updatable or read-only.



Constants in ResultSet

CONCUR_READ_ONLY	Concurrency mode for a ResultSet object that may NOT be updated.
CONCUR_UPDATABLE	Concurrency mode for a ResultSet object that may be updated.
TYPE_FORWARD_ONLY	The type for a ResultSet object whose cursor may move only forward.
TYPE_SCROLL_INSENSITIVE	The type for a ResultSet object that is scrollable but generally ResultSet is insensitive to changes made to the underlying data source while it is open.
TYPE_SCROLL_SENSITIVE	The type for a ResultSet object that is scrollable and reflects changes made to the underlying data source while the result set remains open.

ResultSet Methods

Method	**Meaning**
type get<type> (int index)	Returns the value of the column identified by index.
type get<type> (String name)	Returns the value of the column identified by name.
ResultSetMetaData getMetaData()	Returns metadata of the ResultSet.
boolean next()	Moves cursor to next row and returns true if next row is valid.
boolean wasNull()	Returns true when the last column read was null.
void close()	Closes ResultSet.
boolean absolute(int row)	Moves the cursor to the given row.
int getRow()	Returns current row number. Row number starts with 1.
boolean first ()	Moves to first record. Returns true on success.
boolean last()	Moves to the last record. Returns true on success.

ResultSet Methods

boolean previous()	Moves to the previous record.
void beforeFirst()	Moves to immediately before first record.
void afterLast()	Moves to immediately after the last record.
boolean relative(int)	Moves forward or backward by specified number of rows.
boolean isLast()	Returns true if cursor is on the last record.
boolean isFirst()	Returns true if cursor is on the first record.
boolean isAfterLast()	Returns true if cursor is after the last record.
boolean isBeforeFirst()	Returns true if cursor is before the first record.
void update<type>(int, value)	Changes the existing data.
void cancelRowUpdates()	Cancels the updates made to a row.
void moveToInsertRow()	Creates a new blank row.
void insertRow()	Appends the new row to the table.
void deleteRow()	Deletes the current row of the ResultSet from the table.
void updateRow()	Updates the database with the new contents of the current row.

PreparedStatement Interface

- Contains a precompiled SQL statement. Command can be executed efficiently for multiple times.
- Extends **Statement** interface. Obtained using **prepareStatement()** method of **Connection** interface.

Method	**Meaning**
void clearParameter()	Clears values of all parameters.
void set<type>(int , value)	Sets the given value to the parameter at the specified index.

CallableStatement Interface

- ❑ Used to execute a stored procedure or function.
- ❑ It extends **PreparedStatement** interface.

```
To call a function:
  {?=call<function-name>[<arg1>,<arg2>, ...]}

To call a procedure:
  {call <procedure-name>[<arg1>,<arg2>, ...]}
```

Method	**Meaning**
type get<type>(int index)	Returns the value of the given parameter.
void registerOutParameter (int index, int type)	Registers an OUT parameter at the specified parameter index to the SQLType.
boolean wasNull()	Returns true if last OUT parameter read has SQL NULL.

JDBC Interface vs. Oracle Driver

- ❑ JDBC is a collection of interfaces and a few classes.
- ❑ JDBC driver is a collection of classes that implements JDBC interfaces.
- ❑ For example, JDBC driver for Oracle is a collection of classes provided in a single jar file – **ojdbc6.jar**.
- ❑ The following tables show the relationship between JDBC interface and classes in Oracle driver and MSQL Driver.

JDBC Interface	**Class in Oracle Thin Driver**
Connection	T4CConnection
Statement	OracleStatementWrapper

JDBC Interface	**Class in MySQL Driver**
Connection	ConnectionImpl
Statement	StatementImpl

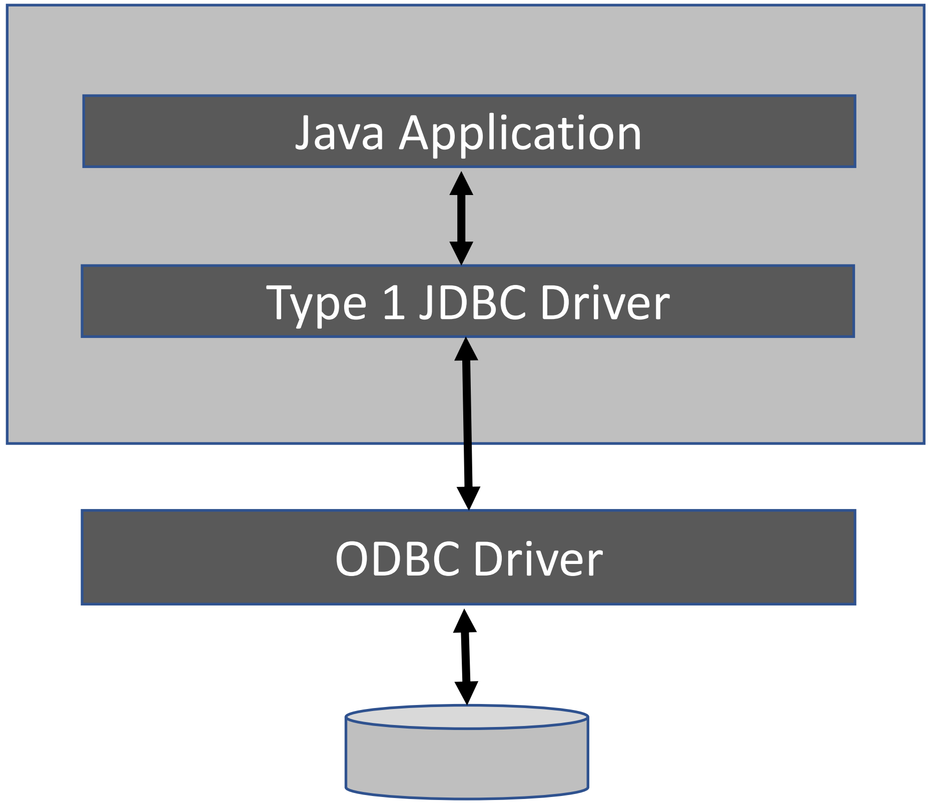
JDBC Drivers

The following are different types of JDBC drivers that can be used with JDBC :

- ❑ JDBC-ODBC Bridge Driver
- ❑ Native API Partly Java Driver
- ❑ Network Protocol Pure Java Driver
- ❑ Native Protocol Pure Java Driver

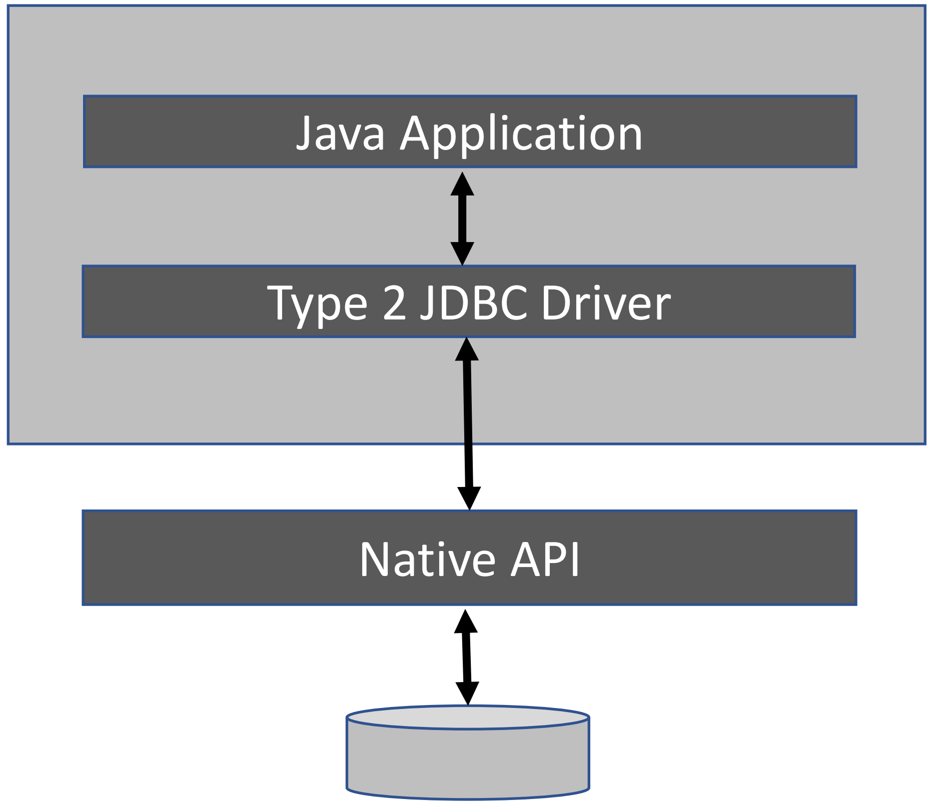
Type 1 : JDBC-ODBC bridge driver

- ❑ Allows you to access any ODBC data source using ODBC Driver.
- ❑ JDBC driver accesses ODBC driver.
- ❑ Requires the ODBC driver to be present on the client.



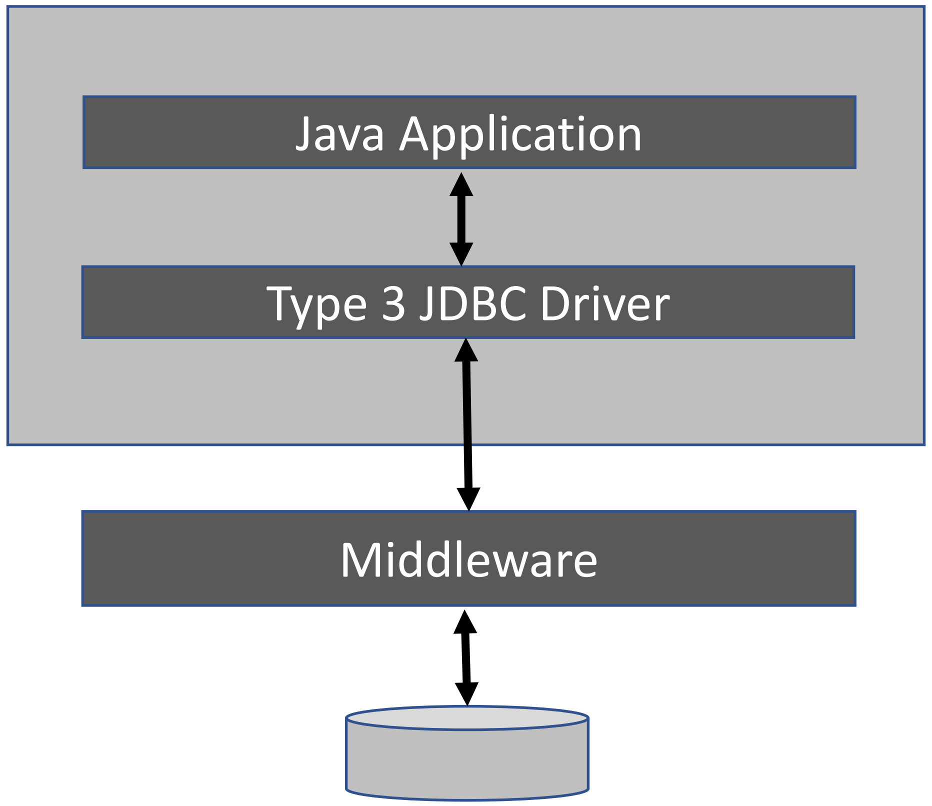
Type 2 : Native API Partly Java Driver

- Converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS.
- Requires that some binary code be loaded on each client machine.
- Oracle’s OCI driver is an example for this.



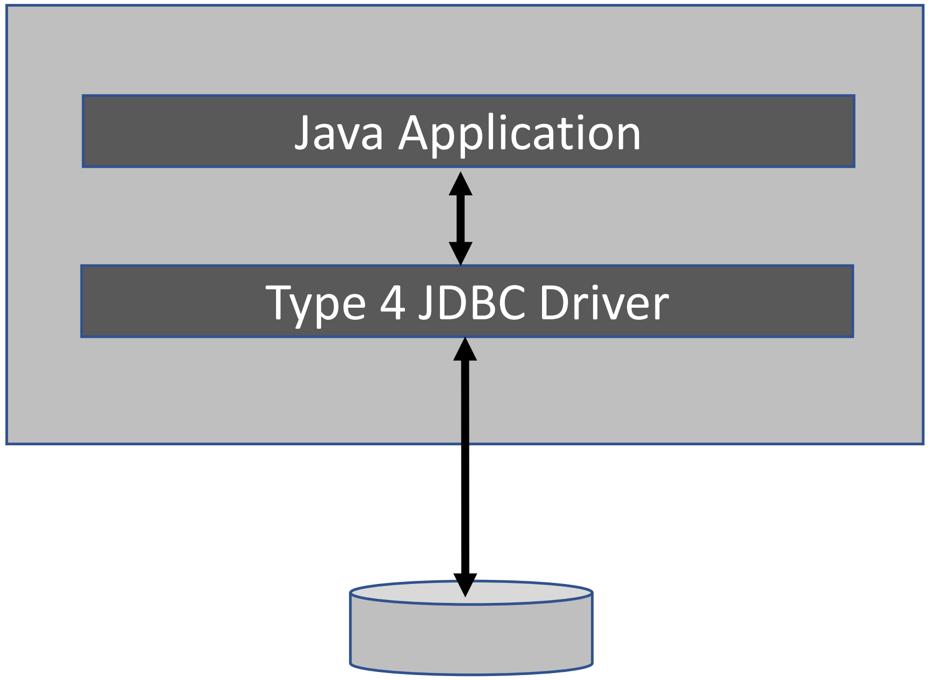
Type 3 : Network Protocol Pure Java Driver

- ❑ Translates JDBC calls into a DBMS-independent net protocol, which is then translated to a DBMS protocol by a server.
- ❑ This driver is provided by middleware (application server) vendor.
- ❑ Fusion Middleware is an example for middleware.
- ❑ Clients need not have any binary code.



Type 4 : Native-protocol Pure Java driver

- This kind of driver converts JDBC calls into the network protocol used by DBMS directly.
- Oracle Thin driver is an example for this.



Note: Eventually, driver categories 3 and 4 will be the preferred way to access databases from JDBC.

Transaction Management

- ❑ A transaction consists of one or more statements that have been executed, completed, and then either committed or rolled back. When the method commit or rollback is called, the current transaction ends and another one begins.
- ❑ A new connection is in auto-commit mode by default, meaning that when a statement is completed, the method commit will be called on that statement automatically.

Method	**Meaning**
void setAutoCommit(boolean)	Turns on/off auto commit mode.
boolean getAutoCommit()	Returns the current auto commit mode.
void commit()	Makes all changes since previous commit point permanent.
void rollback()	Drops all changes made since the previous commit/rollback.
void rollback(Savepoint)	Rolls back changes made from the given savepoint.
Savepoint setSavepoint(name)	Sets a savepoint in the current transaction.
void releaseSavepoint(sp)	Releases savepoint from current transaction.

Transaction Example

```
con.setAutoCommit(false);
try {
    st.executeUpdate(cmd1);
    st.executeUpdate(cmd2);
    con.commit();
}
catch(Exception ex) {
    con.rollback();
}
```

Transaction with Savepoint

- Savepoint is a point within a transaction to which you can roll back. Connection interface provides **setSavepoint** () method to set savepoint.
- Method **releaseSavepoint**(savepoint) releases savepoint from the current transaction.

```
Statement stmt = conn.createStatement();
int rows = stmt.executeUpdate( sqlcommand );
Savepoint savepoint1 = conn.setSavepoint("SAVEPOINT1");
rows = stmt.executeUpdate(sqlcommand);
...
if ( rows > 5 )
   conn.rollback(svpt1);
...
conn.commit();
```

Batch Update

- ❑ It allows multiple DML statements to be executed in a single request.
- ❑ Reduces overhead on database server as it has to execute only one command.
- ❑ Improves performance when large number of statements are executed.

```
Statement stmt = con.createStatement();
try {
   stmt.addBatch
    ("update employees set salary=salary+2000 where salary>10000");
   stmt.addBatch
    ("update employees set salary=salary+1000 where salary<10000");

   int[] uc = stmt.executeBatch();//execute batch
   con.commit();
}
```