JAVA FULL STACK DEVELOPMENT PROGRAM

Session 8: JDBC

OUTLINE

- Introduction to JDBC
- Database handling using JDBC
  - Load the Driver
  - Establish a Connection
  - SQL Statements
  - Execution
- Transaction Management in JDBC
- Exception Handling in JDBC
- Batch Processing in JDBC
- JDBC Advance

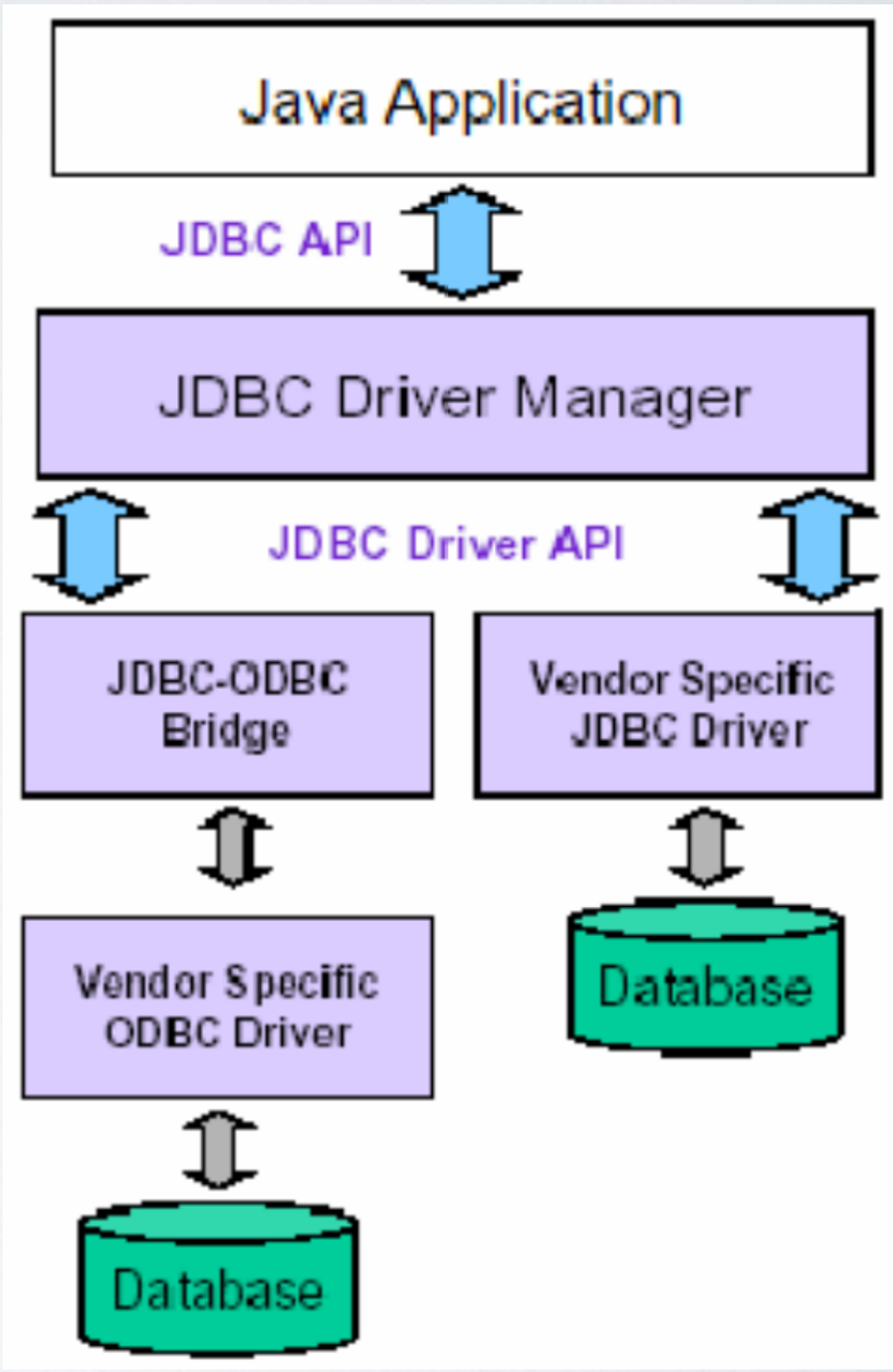
INTRODUCTION TO JDBC

- JDBC stands for "Java Database Connectivity".
- It is an API (Application Programming Interface) which consists of a set of Java classes, interfaces and exceptions and a specification to which both JDBC driver vendors and JDBC developers (like we) adhere when developing applications.
- Now we are only focusing on RDBMS (Most Java positions are related to RDBMS, non relational db is a plus)

JDBC ARCHITECTURE

- What design pattern is implied in this architecture?

- What does it buy for us?



DATABASE HANDLING USING JDBC

- JDBC stands for Java database connectivity.
- A standard API for all Java programs to connect to databases. The JDBC API is available in two packages:
  - Core API java.sql.
  - Standard extension to JDBC API javax.sql (supports connection pooling, transactions, etc.)
- JDBC defines a few steps to connect to a database and retrieve/insert/update databases.
  - Load the driver
  - Establish connection
  - Create statements
  - Execute query and obtain result
  - Iterate through the results
  - Close connection

JDBC DRIVER

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

There are 4 types of JDBC drivers: (Only need to know)

- JDBC-ODBC bridge driver
- Native-API driver (partially java driver)
- Network Protocol driver (fully java driver)
- Thin driver (fully java driver)

LOAD THE DRIVER

- The driver is loaded with the help of a static method,
  - Class.forName(drivername)

- Every database has its own driver.

Database name	Driver Name
MS Access	sun.jdbc.odbc.JdbcOdbcDriver
Oracle	oracle.jdbc.driver.OracleDriver
Microsoft SQL Server 2000 (Microsoft Driver)	com.microsoft.sqlserver.jdbc.SQLServerDriver
MySQL (MM.MySQL Driver)	org.gjt.mm.mysql.Driver

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.
- Advantages:
  - It is independent of database systems and operating system
    - 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. (the computer having the driver)

JDBC-ODBC BRIDGE DRIVER

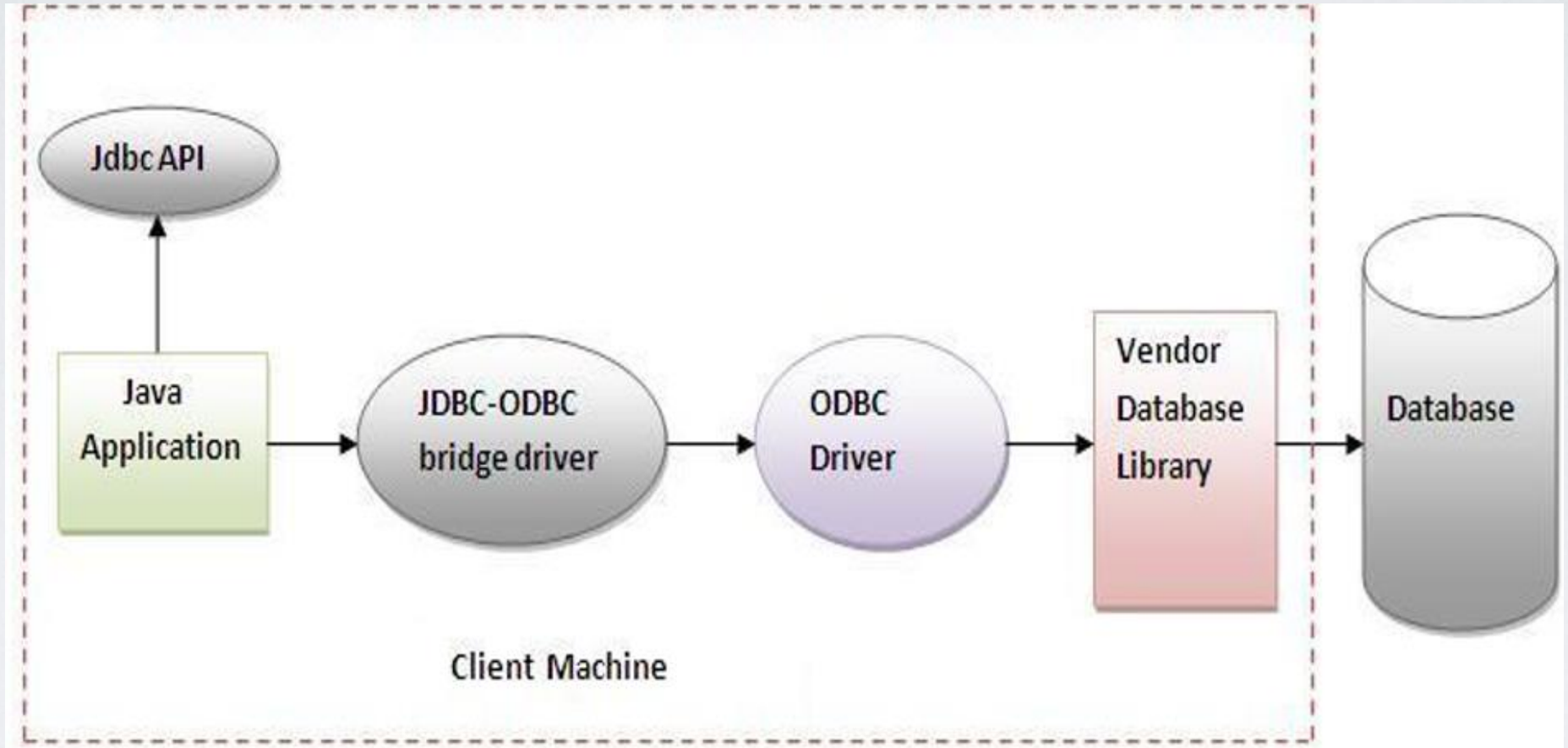


Figure- JDBC-ODBC Bridge Driver

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.
- 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.

NATIVE-API DRIVER

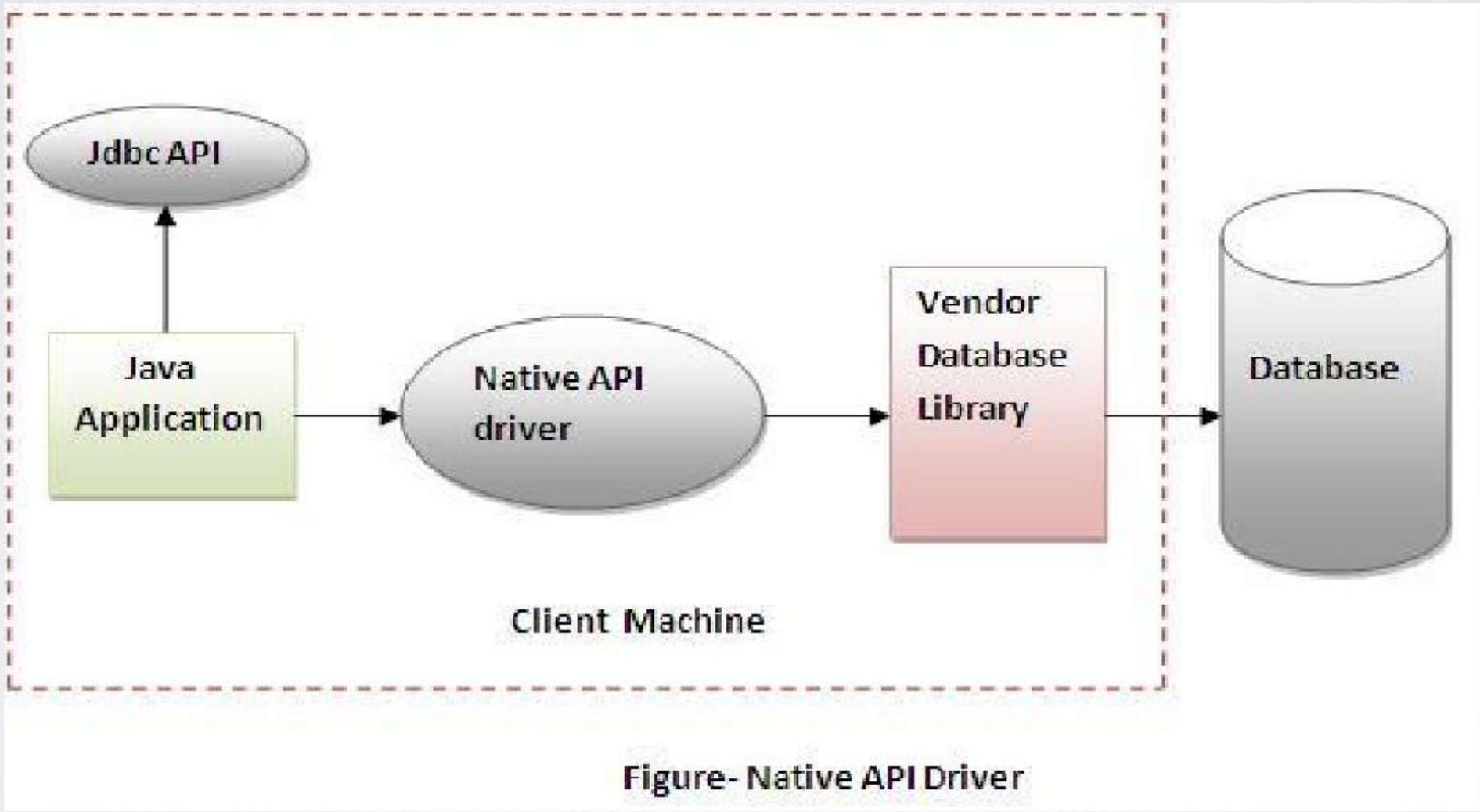


Figure- Native API Driver

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.
- 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.

NETWORK PROTOCOL DRIVER

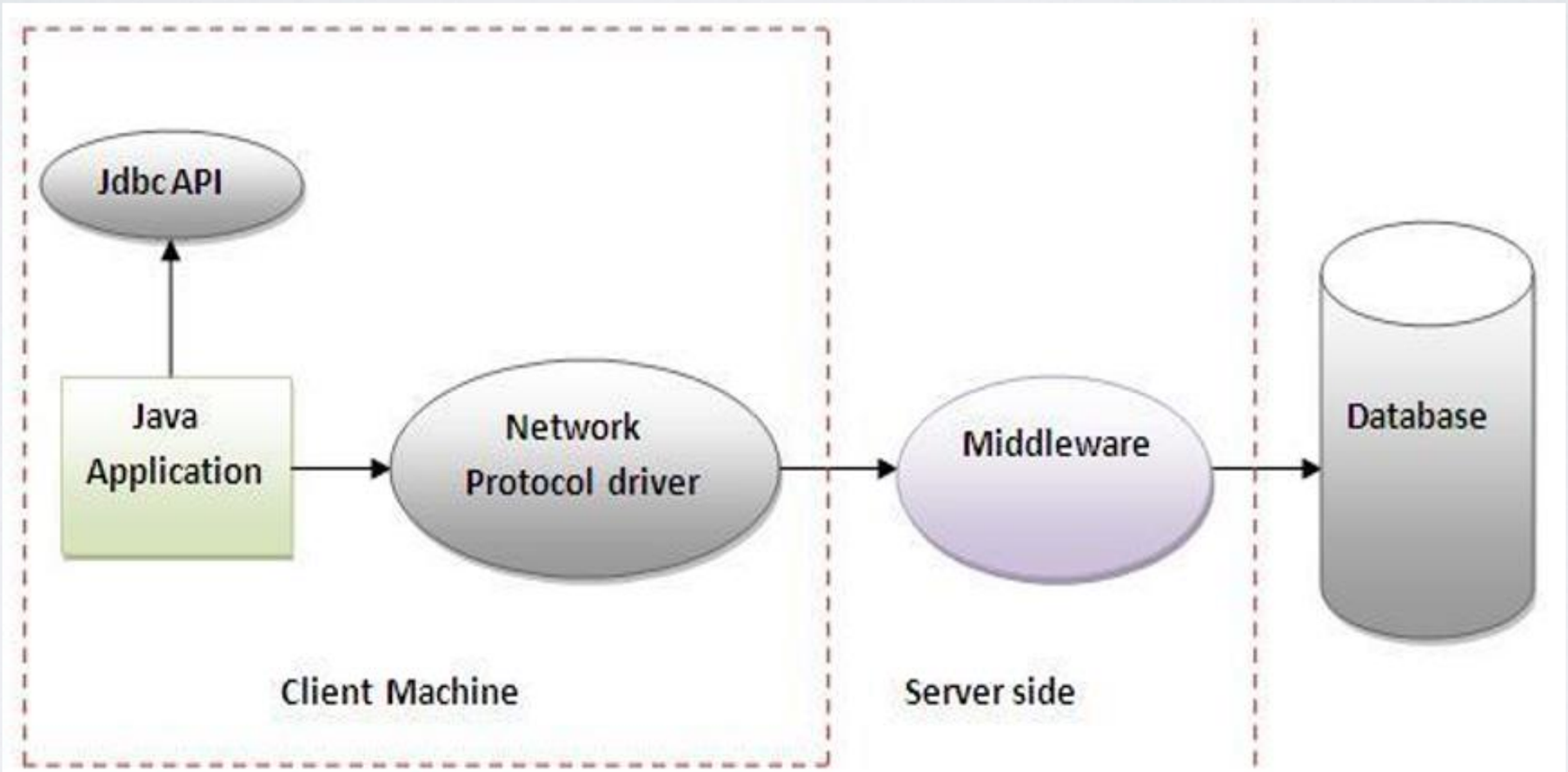


Figure- Network Protocol Driver

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.
- Advantage:
  - Better performance than all other drivers.
  - No software is required at client side or server side.
- Disadvantage:
  - Drivers depends on the Database.

THIN DRIVER

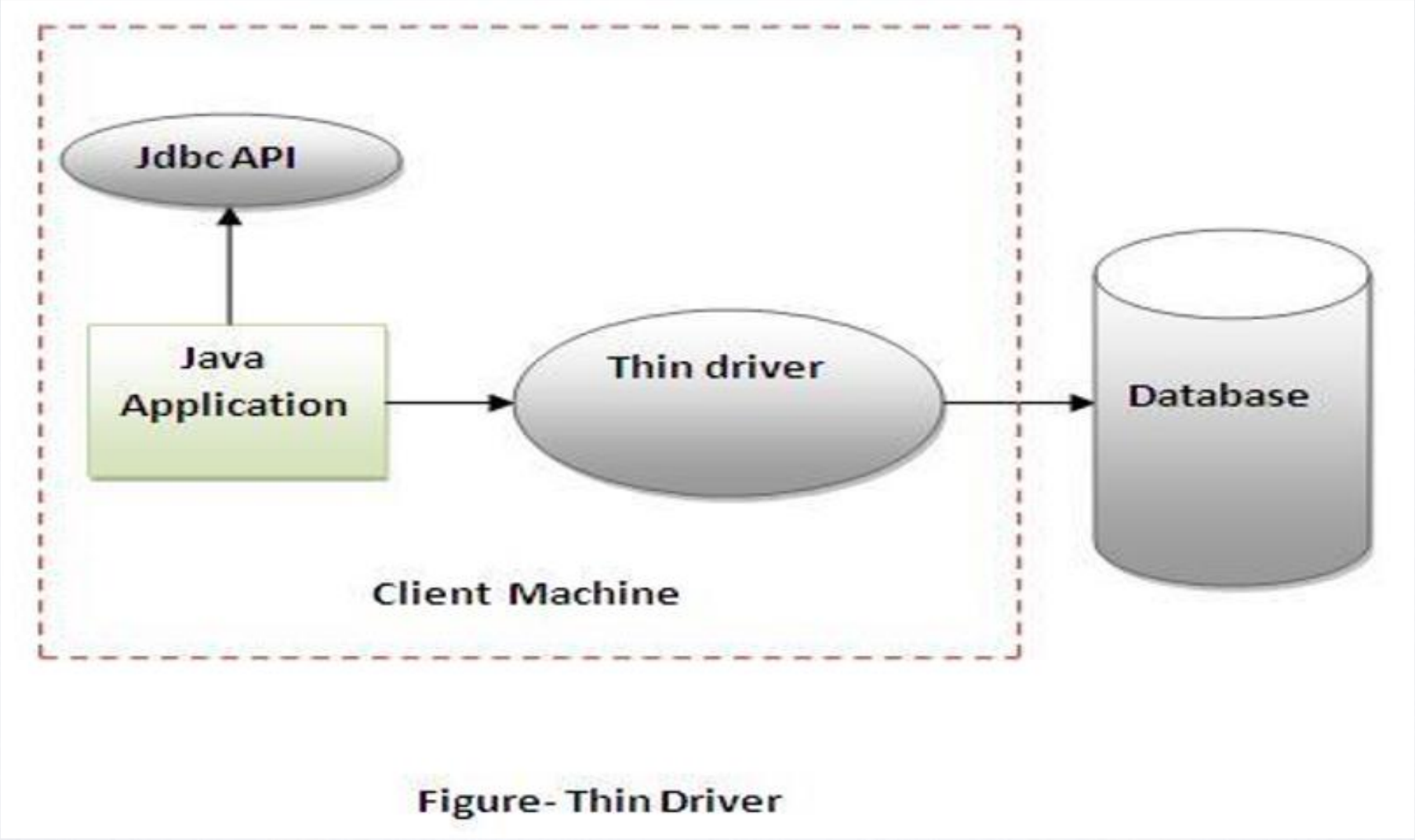
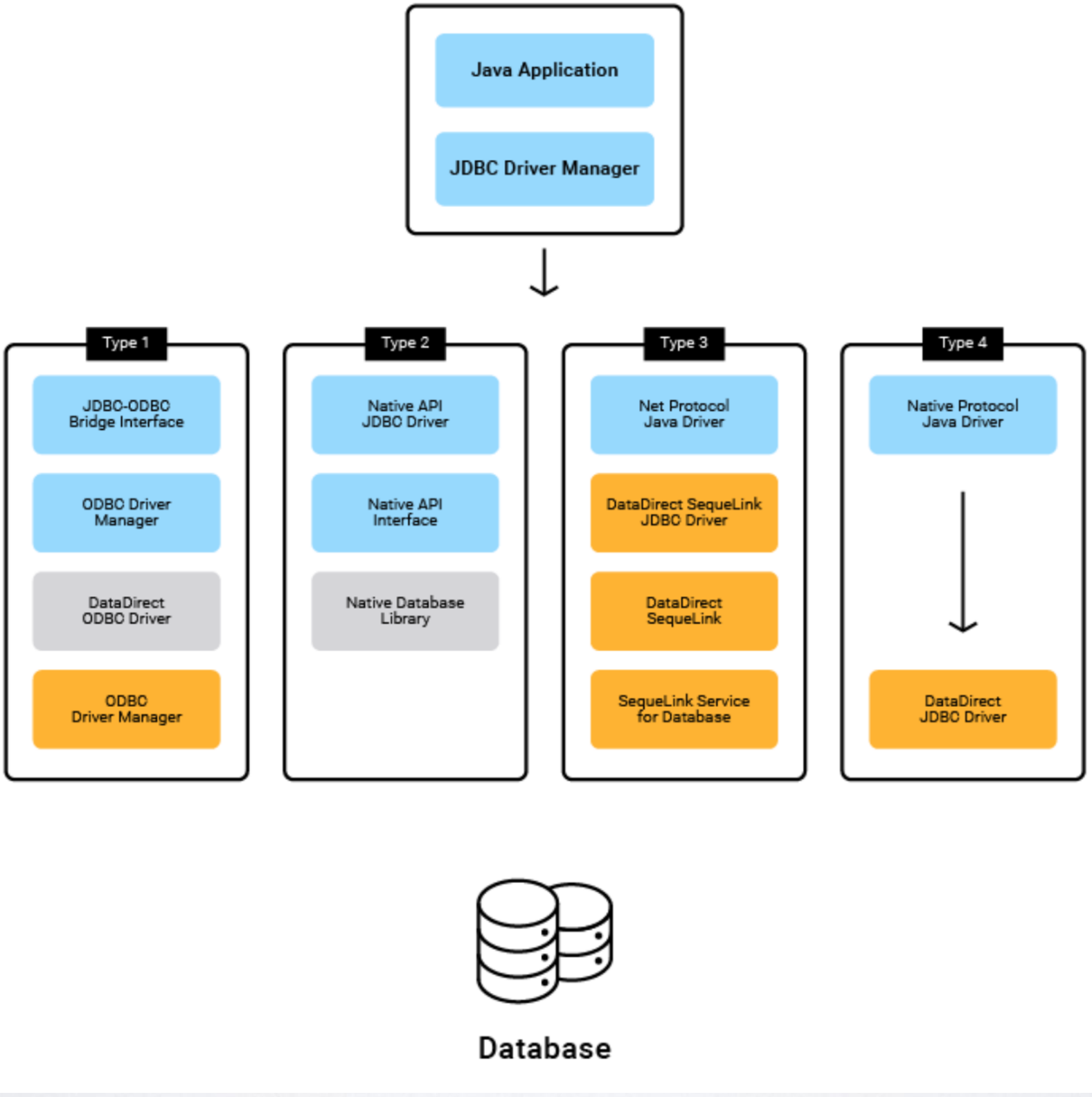


Figure- Thin Driver


Java Application
JDBC Driver Manager
Type 1
JDBC-ODBC Bridge Interface
ODBC Driver Manager
DataDirect ODBC Driver
ODBC Driver Manager
Type 2
Native API JDBC Driver
Native API Interface
Native Database Library
Type 3
Net Protocol Java Driver
DataDirect SequeLink JDBC Driver
DataDirect SequeLink
SequeLink Service for Database
Type 4
Native Protocol Java Driver
DataDirect JDBC Driver
Database

ESTABLISH A CONNECTION

- A connection to the database is established using the static method *getConnection(databaseUrl)* of the DriverManager class.
- The DriverManager class is class for managing JDBC drivers.
- The database URL takes the following shape jdbc:subprotocol:subname.
- If any problem occurs during accessing the database, an SQLException is generated, else a Connection object is returned which refers to a connection to a database.
- Connection is actually an interface in java.sql package.
  - *Connection con=DriverManager.getConnection(databaseUrl);*

FEW DATABASE URLS

Database	Database URL
MS Access	jdbc:odbc:<DSN>
Oracle thin driver	jdbc:oracle:thin:@<HOST>:<PORT>:<SID>
Microsoft SQL Server 2000	jdbc:microsoft:sqlserver://<HOST>:<PORT>[;DatabaseName=<DB>]
MySQL (MM.MySQL Driver)	jdbc:mysql://<HOST>:<PORT>/<DB>

CREATE STATEMENT

- The connection is used to send SQL statements to the database.
- Three interfaces are used for sending SQL statements to databases
  - Statement
  - PreparedStatement
  - Callable Statement
- Three methods of the Connection object are used to return objects of these three statements.
- A Statement object is used to send a simple SQL statement to the database with no parameters.
  - *Statement stmt = con.createStatement();*

CREATE STATEMENT

- Statement
  - If n rows need to be inserted, then the same statement gets compiled n number of times.
- So to increase efficiency, we use precompiled PreparedStatement.
- A PreparedStatement object sends pre-compiled statements to the databases with or without IN parameters.
  - Only the values that have to be inserted are sent to the database again and again.
    - *PreparedStatement ps = con.prepareStatement(String query);*
  - A CallableStatement object is used to call stored procedures.
    - *CallableStatement cs = con.prepareCall(String query);*

EXECUTE QUERY

- Two methods are used
  - ResultSet executeQuery(String sqlQuery) throws SQLException
  - int executeUpdate(String sqlQuery) throws SQLException
  - boolean execute(String sqlQuery) throw SQLException
- executeQuery is used for executing SQL statements that return a single ResultSet, e.g. a select statement.
  - The rows fetched from database are returned as a single ResultSet object. For example,
  - *ResultSet rs = stmt.executeQuery("select * from emp");*
- executeUpdate is used for DDL and DML SQL statements like insert, update, delete, and create.
  - returns an integer value for DML to indicate the number of rows affected and 0 for DDL statements which do not return anything.

EXECUTE QUERY

- Prepared Statement
  - PreparedStatement ps = con.prepareStatement(“update emp set salary=? where empid=?”);
- The statement is sent to database and is prepared for execution, only the value of the IN (?) parameters need to be sent.
  - ps.setInt(1,100000);
  - ps.setString(2,”Emp001”);
  - ps.executeUpdate();
- The *execute* method is used when the statement may return more than one ResultSet or update counts or a combination of both.
  - boolean isResultSet = stmt.execute("select 'Hello '||USER from dual")
  - if (isResultSet) {

```
        rslt = stmt.getResultSet(  );
    }
```

ITERATE RESULT SET

Iterate ResultSet

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

MAPPING TYPES JDBC - JAVA

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()

DATABASE METADATA

- *Metadata* basically means the data that provide a structured description about some other data.
- From a programmer's point of view, database metadata refers to data about database data or, more elaborately, the information about tables, views, column types, column names, result sets, stored procedures, and databases.
- Use of JDBC Metadata API
  - Database users, tables, views, stored procedures
  - Database schema and catalog information
  - Table, view, column privileges
  - Information about primary key, foreign key of a table

RESULT SET META DATA

```
ResultSetMetaData rsmd=rs.getMetaData();
  System.out.println("Column in ResultSet:"+rsmd.getColumnCount());
for(int i=1;i<=rsmd.getColumnCount();i++)
{
System.out.println("Column Name :"+rsmd.getColumnName(i));
System.out.println("Column Type :"+rsmd.getColumnTypeName (i));
}
```

CLOSE CONNECTION

- Just like file I/O, we have to close the database connection after we finish all the jobs.

- *statement.close()*

- *connection.close()*

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

- JDBC allows SQL statements to be grouped together into a single transaction
- Transaction control is performed by the Connection object,
  - default mode is auto-commit, I.e., each sql statement is treated as a transaction
- We can turn off the auto-commit mode with con.setAutoCommit(false);
- And turn it back on with con.setAutoCommit(true);
- Once auto-commit is off, no SQL statement will be committed until an explicit is invoked con.commit();
- At this point all changes done by the SQL statements will be made permanent in the database.

TRANSACTION MANAGEMENT IN JDBC

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

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

EXCEPTION HANDLING IN JDBC

- Programs should recover and leave the database in a consistent state.

- If a statement in the try block throws an exception or warning, it can be caught in one of the corresponding catch statements

- How might a finally {…} block be helpful here?

  - E.g., you could rollback your transaction in a catch { …} block or close database connection and free database related resources in finally {…} block

BATCH PROCESSING IN JDBC

- Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast.

- The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.

- Advantage of Batch Processing

    - Fast Performance

EXAMPLE OF BATCH PROCESSING USING PREPAREDSTATEMENT

```
import java.sql.*;

class FetchRecords{

  public static void main(String args[])throws Exception{

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

     Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:x e","system",“xe");

     con.setAutoCommit(false);

     Statement stmt=con.createStatement();

     stmt.addBatch("insert into user420 values(190,'abhi',40000)");

     stmt.addBatch("insert into user420 values(191,'umesh',50000)");

     stmt.executeBatch();//executingthe batch

     con.commit();

     con.close();

}}
```

JAVA CALLABLESTATEMENT INTERFACE

- CallableStatement interface is used to call the stored procedures and functions.

- We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

- Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

JAVA CALLABLESTATEMENT INTERFACE

```
create or replace
PROCEDURE getEmployee
(in_id IN EMPLOYEE.EMPID%TYPE,
 out_name OUT EMPLOYEE.NAME%TYPE,
 out_role OUT EMPLOYEE.ROLE%TYPE,
 out_city OUT EMPLOYEE.CITY%TYPE,
 out_country OUT EMPLOYEE.COUNTRY%TYPE
 )
AS
BEGIN
  SELECT NAME, ROLE, CITY, COUNTRY
  INTO out_name, out_role, out_city, out_country
  FROM EMPLOYEE
  WHERE EMPID = in_id;

END;
```

```
con = DBConnection.getConnection();
stmt = con.prepareCall("{call getEmployee(?,?,?,?,?)}");
stmt.setInt(1, id);

//register the OUT parameter before calling the stored procedure
stmt.registerOutParameter(2, java.sql.Types.VARCHAR);
stmt.registerOutParameter(3, java.sql.Types.VARCHAR);
stmt.registerOutParameter(4, java.sql.Types.VARCHAR);
stmt.registerOutParameter(5, java.sql.Types.VARCHAR);

stmt.execute();

//read the OUT parameter now
String name = stmt.getString(2);
String role = stmt.getString(3);
String city = stmt.getString(4);
String country = stmt.getString(5);
```

SCROLLABLE RESULT SET

- A scrollable ResultSet is one which allows us to retrieve the data in forward direction as well as backward direction but no updating is allowed

- In order to make the non-scrollable ResultSet as scrollable ResultSet we must use the following createStatement() method which is present in Connection interface.

- public Statement createStatement(int Type, int Mode);
  - TYPE_FORWARD_ONLY -> 1
  - TYPE_SCROLL_INSENSITIVE -> 2
  - CONCUR_READ_ONLY -> 3

SCROLLABLE RESULT SET

- Create a scrollable result set:
  - *Statement st = con.createStatement (ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);*
  - *ResultSet rs = st.executeQuery ("select * from empleyee");*
  - *rs.previous(); // go back in the RS*
  - *rs.relative(-5); // go 5 records back*
  - *rs.relative(7); // go 7 records forward*
  - *rs.absolute(100); // go to 100th record*

UPDATABLE RESULT SET

- An updatable ResultSet object allows us to update a column value, insert column values and delete a row

  - *con.createStatement(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);*

- *ResultSet rs = st.executeQuery ("select * from empleyee");*

- *rs.updateInt("grade", grade+10);*

- *rs.updateRow();*

ANY QUESTIONS?