Database Programming

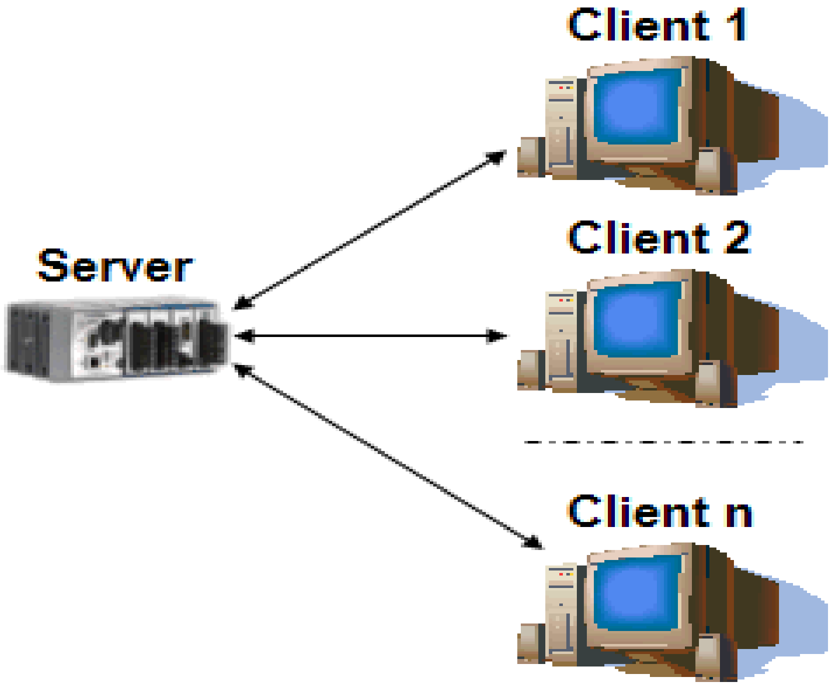
- **JDBC:** Introduction
- JDBC Architecture
- Types of Drivers
- Statement
- ResultSet, Read Only ResultSet, Updatable ResultSet, Forward Only ResultSet, Scrollable ResultSet
- PreparedStatement, Connection Modes, SavePoint, Batch Updations
- CallableStatement, BLOB & CLOB

Database Architectures

- **Two-tier**
- **Three-tier**
- **N-tier**

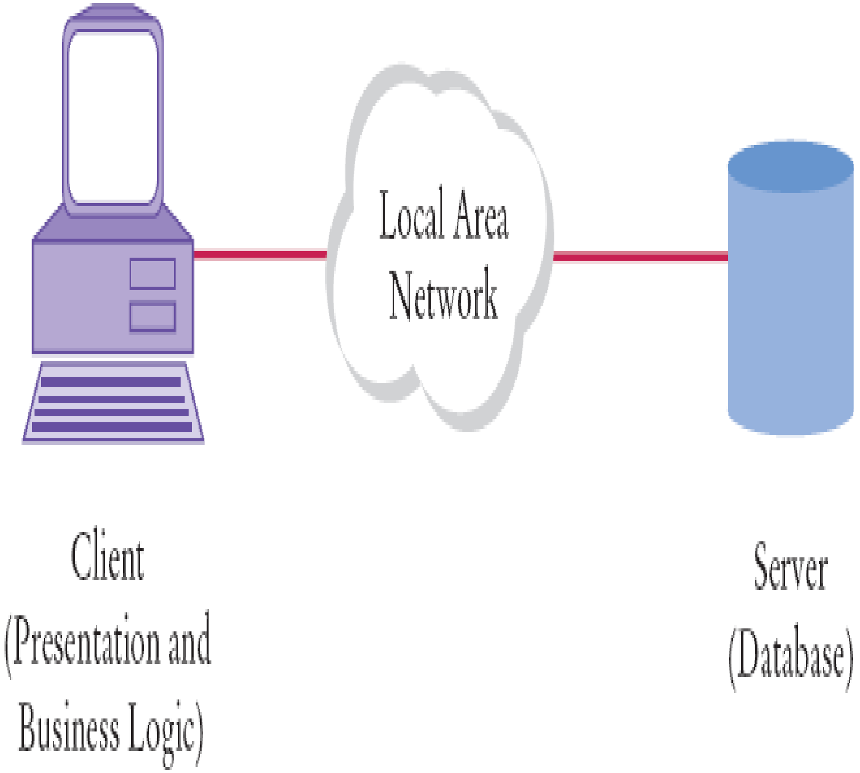
Two-Tier Architecture

- **Client connects directly to server**
- **e.g. HTTP, email**



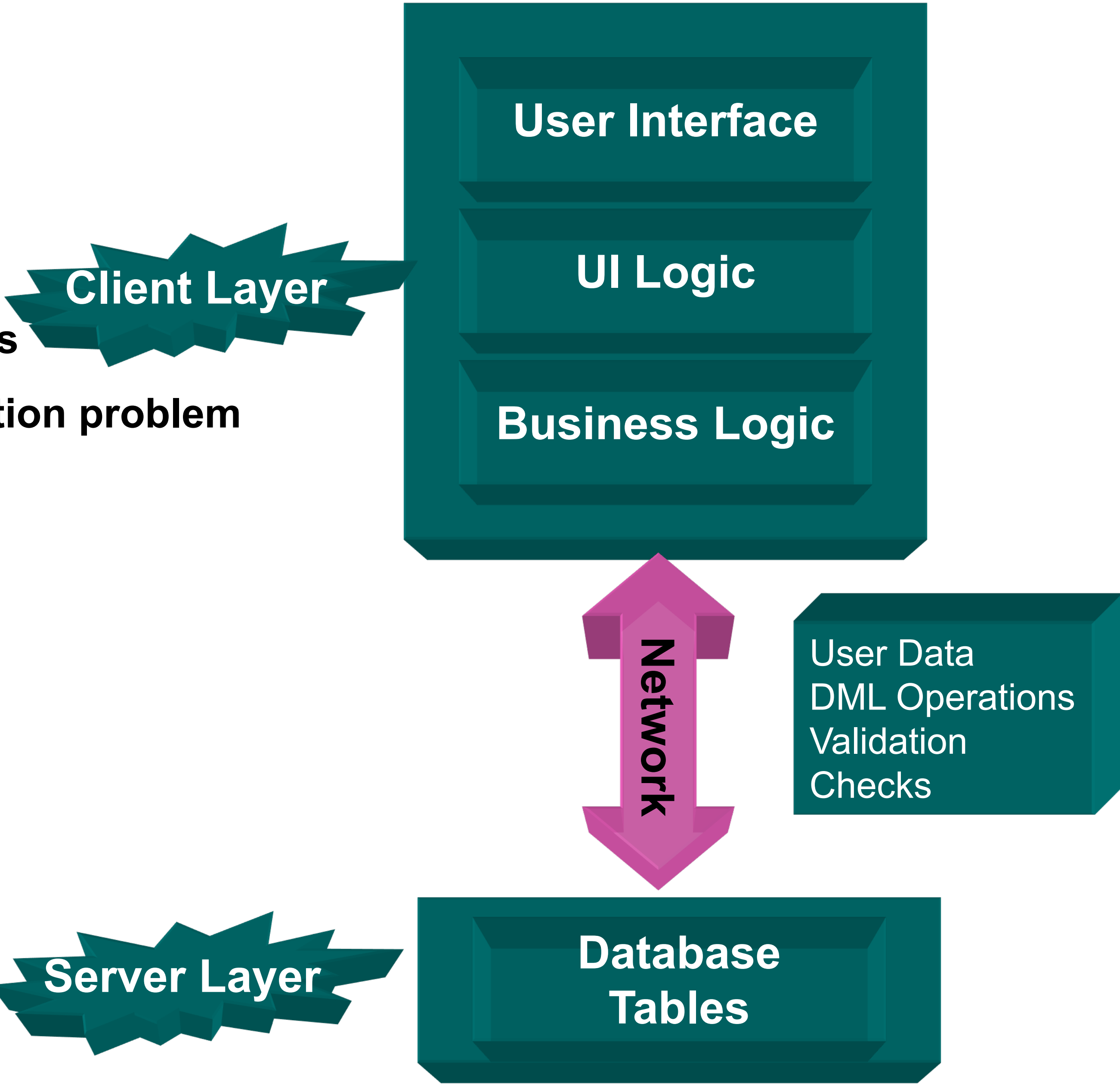
Two-Tier Pros

- **Simple**
- **Client-side scripting offloads work onto the client**



Two-Tier Cons

- **Fat client**
- **Server bottlenecks**
- **Software Distribution problem**
- **Inflexible**



Client Layer

User Interface

UI Logic

Business Logic

Network

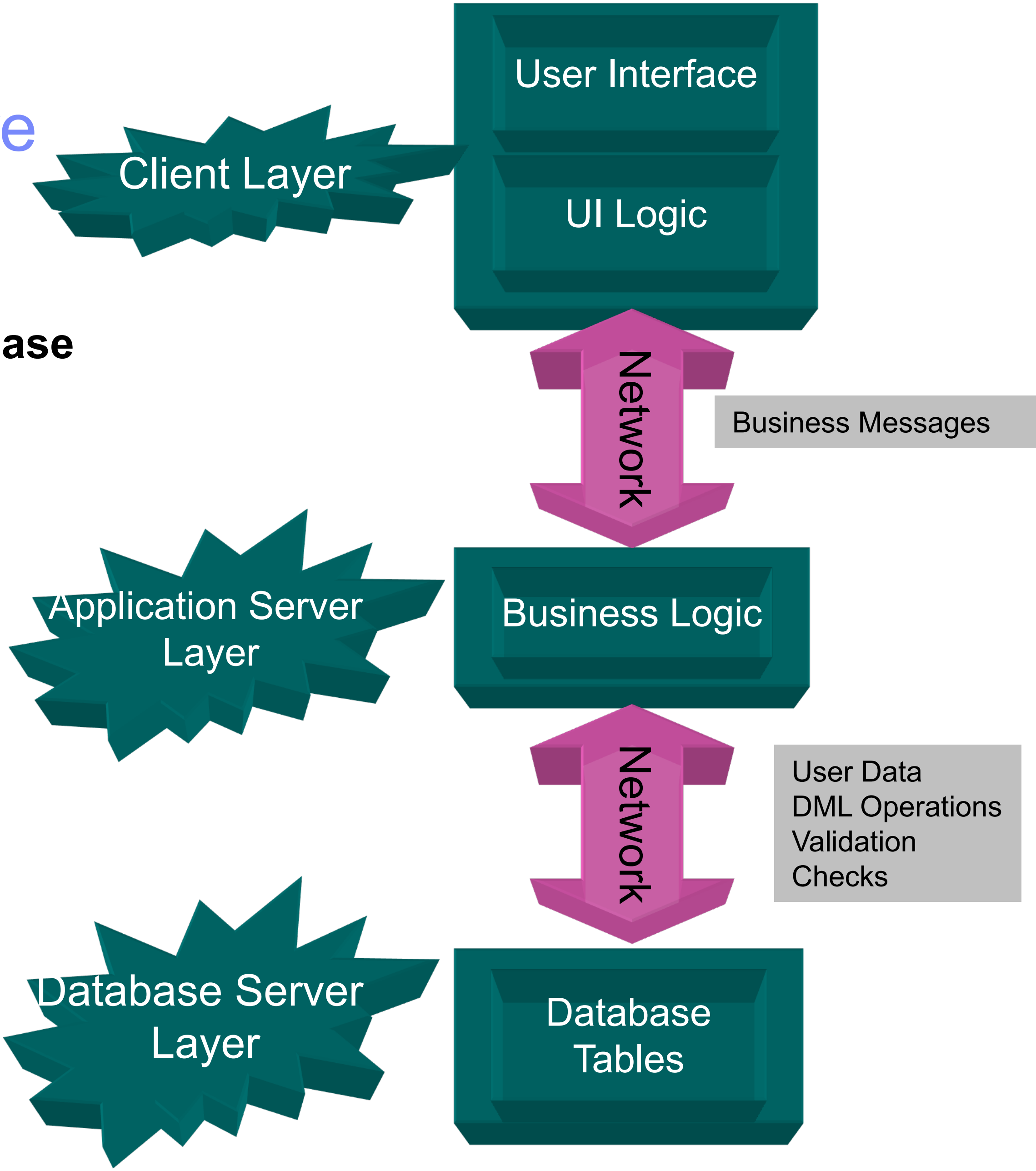
User Data
DML Operations
Validation
Checks

Server Layer

Database Tables

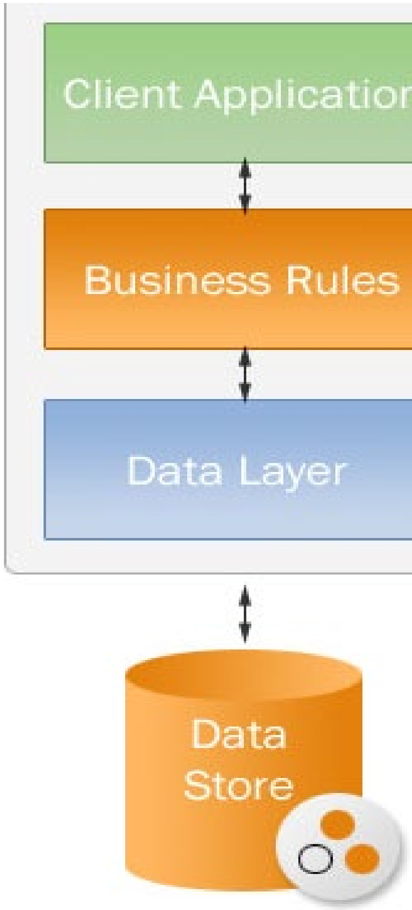
Three-Tier Architecture

- **Application Server sits between client and database**



Three-Tier Pros

- **flexible: can change one part without affecting others**
- **can connect to different databases without changing code**
- **specialization: presentation / business logic / data management**
- **can cache queries**

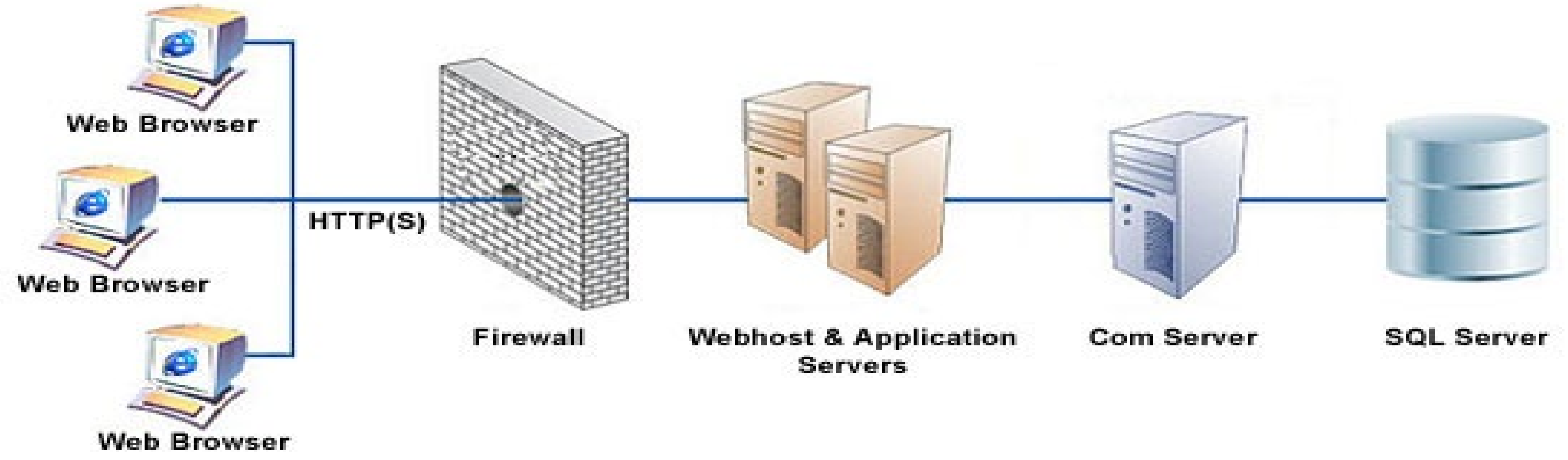


Three-Tier Cons

- **higher complexity**
- **higher maintenance**
- **lower network efficiency**
- **more parts to configure (and buy)**

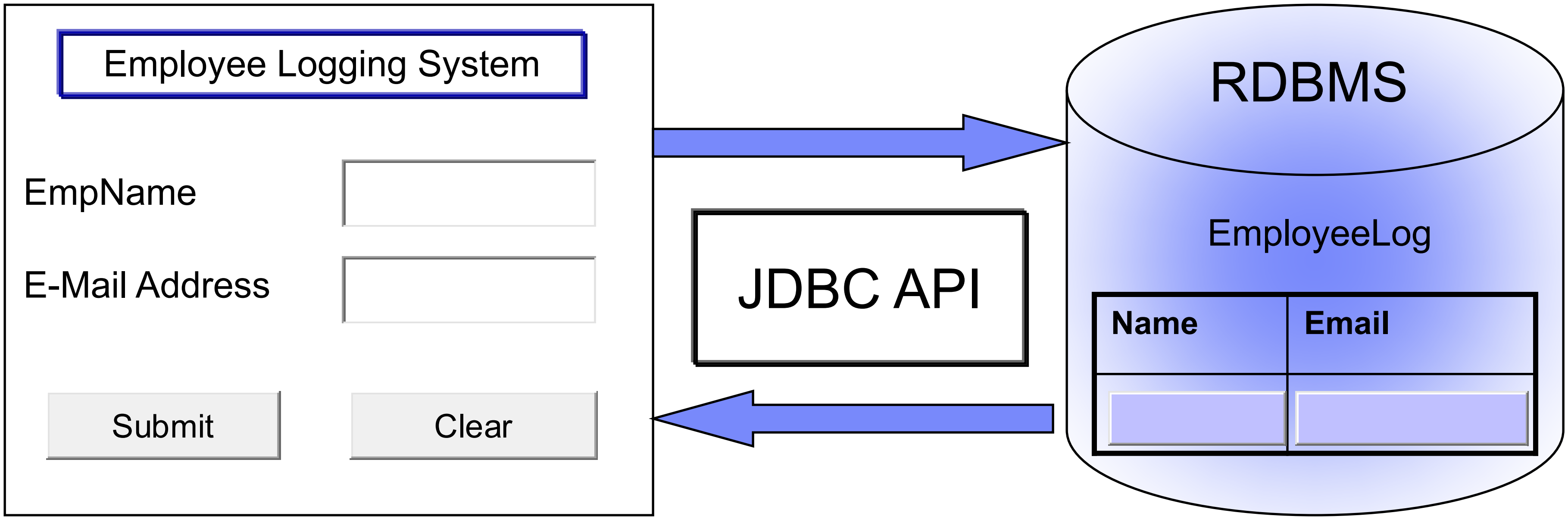
N-Tier Architecture

- **Design your application using as many “tiers” as you need**
- **Use Object-Oriented Design techniques**
- **Put the various components on whatever host makes sense**
- **Java allows N-Tier Architecture, especially with RMI and JDBC**



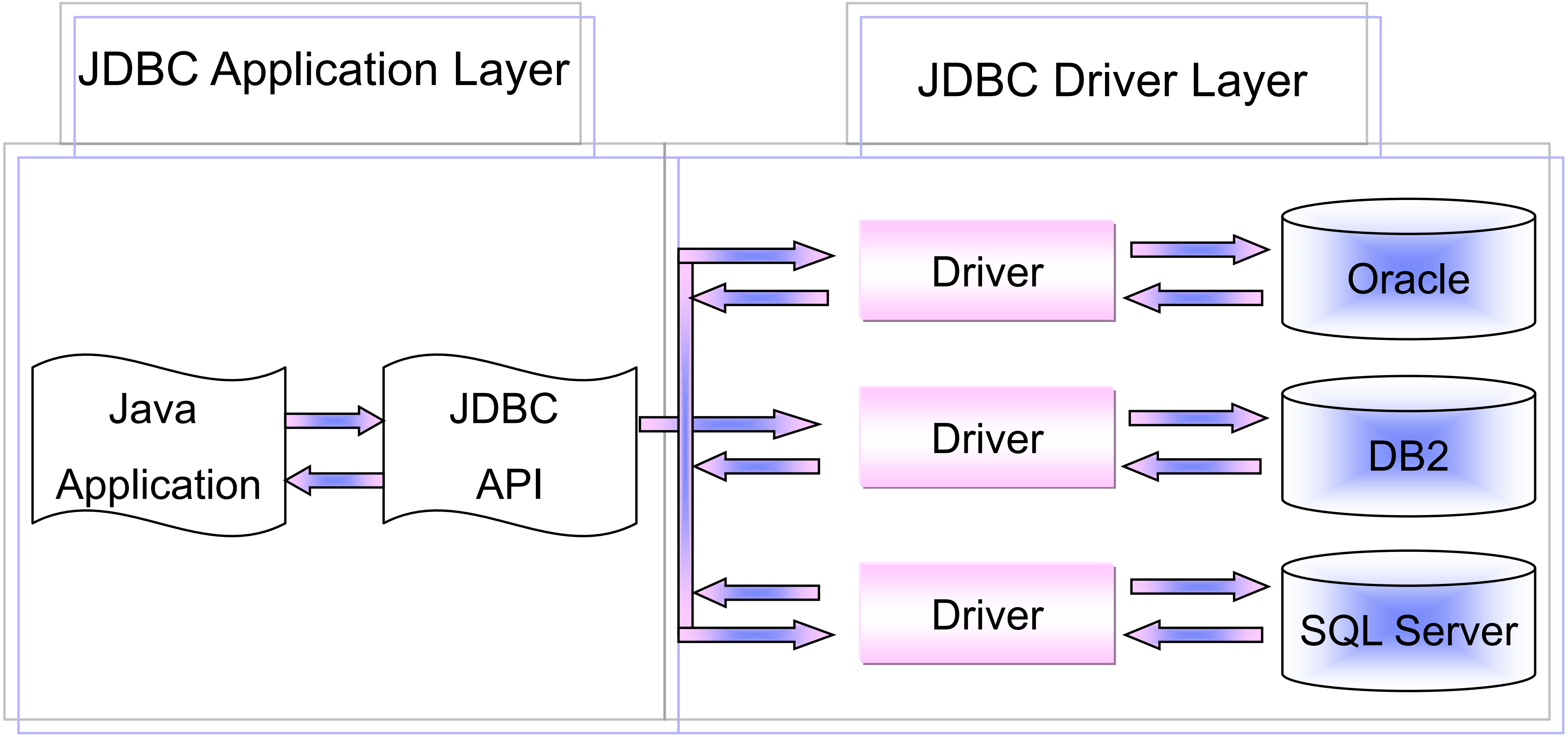
Java Application $\xleftrightarrow{\text{Connects to}}$ Database

- The below given figure shows the Employee Logging System application developed in Java interacting with the Employee database using the JDBC API:

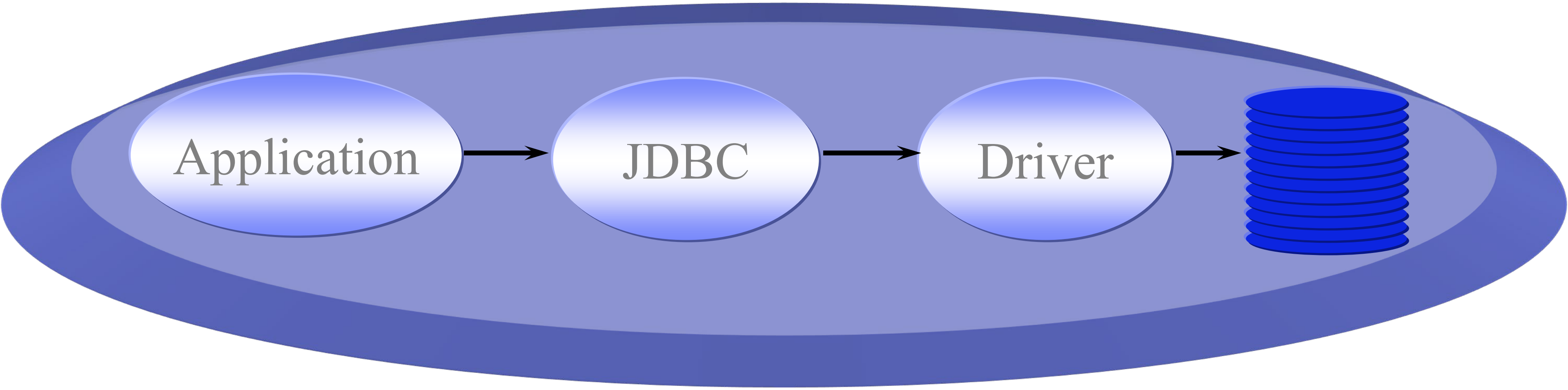


JDBC Architecture

- It can be categorized into into two layers:



JDBC Architecture (Continued)



- **Java code calls JDBC library**
- **JDBC loads a *driver***
- **Driver talks to a particular database**
- **Can have more than one driver -> more than one database**
- **Can change database engines without changing any application code**

JDBC Drivers

- **Type I: “Bridge” -**

JDBC-ODBC Bridge Driver

- **Type II: “Native” -**

Native-API Partly-Java Driver

- **Type III: “Middleware” -**

JDBC-Net Pure-Java Driver

- **Type IV: “Pure” -**

Native Protocol Pure-Java Driver

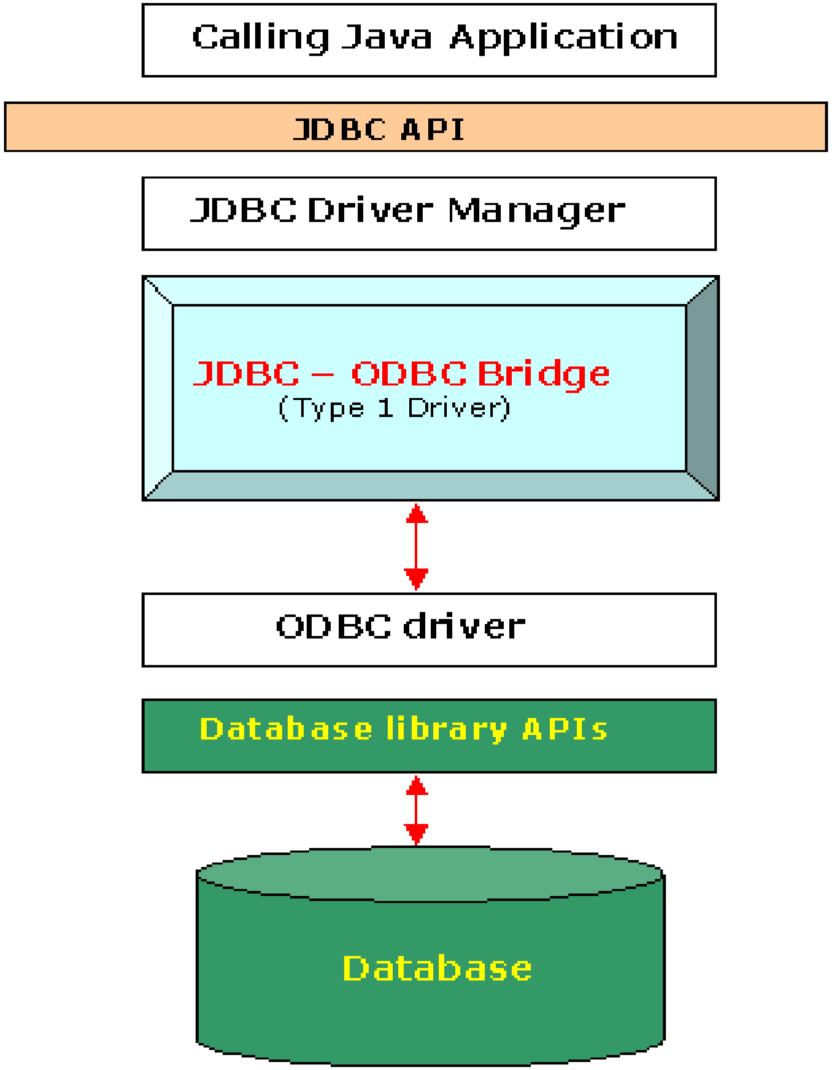
Overview of All Drivers

Type I Drivers

- **Use bridging technology**
- **Translates query obtained by JDBC into corresponding ODBC query, which is then handled by the ODBC driver.**
- **Almost any database for which ODBC driver is installed, can be accessed.**

Go Back To

JDBC Driver List



Disadvantage of Type-I Driver

- **Performance overhead since the calls have to go through the JDBC overhead bridge to the ODBC driver, then to the native db connectivity interface.**
- **The ODBC driver needs to be installed on the client machine.**
- **Not good for Web**

Go Back To

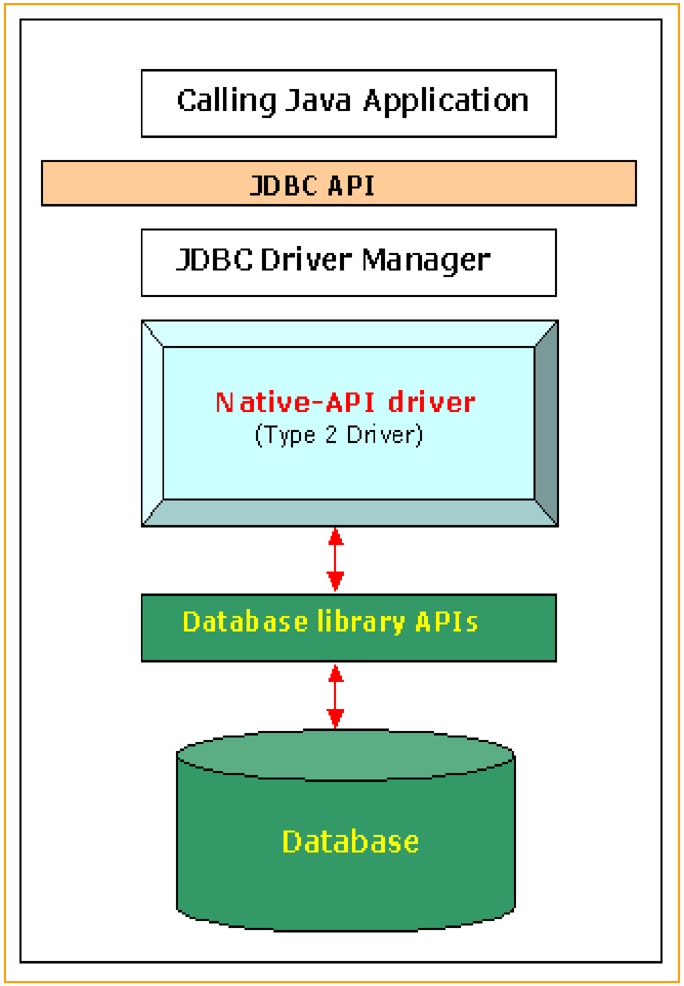
JDBC Driver List

Type II Drivers

- **Native API drivers**
- **Better performance than Type 1 since no jdbc to odbc translation is needed.**
- **Converts JDBC calls into calls to the client API for that database.**

Go Back To

JDBC Driver List



Disadvantage of Type-II Driver

- **The vendor client library needs to be installed on the client machine.**
- **Cannot be used in internet due the client side software needed.**
- **The driver is compiled for use with the particular operating system.**
- **Mostly obsolete now**
- **Not good for Web**

Go Back To

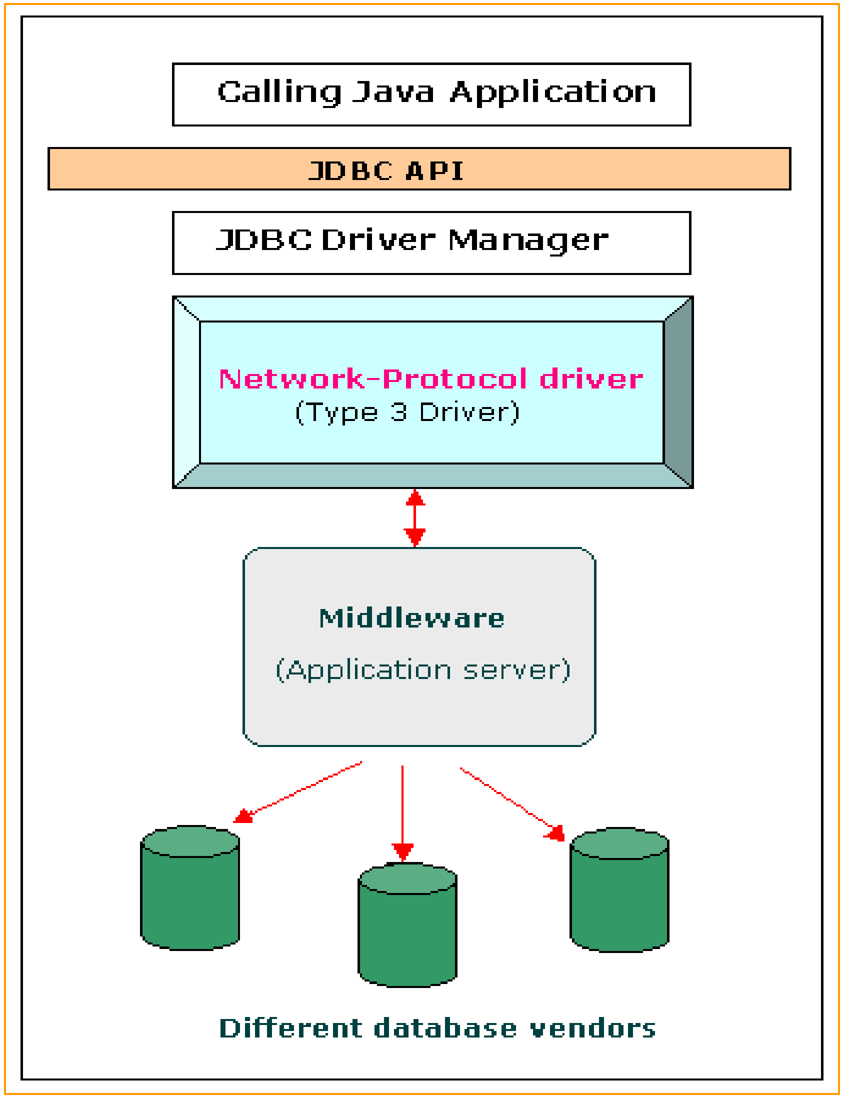
JDBC Driver List

Type III Drivers

- **Follows a three tier communication approach.**
- **Calls middleware server, usually on database host**
- **Very flexible -- allows access to multiple databases using one driver**
- **Only need to download one driver**

Go Back To

JDBC Driver List



Disadvantage of Type-III Driver

- **Requires database-specific coding to be done in the middle tier.**
- **An extra layer added may result in a time-bottleneck.**

Go Back To

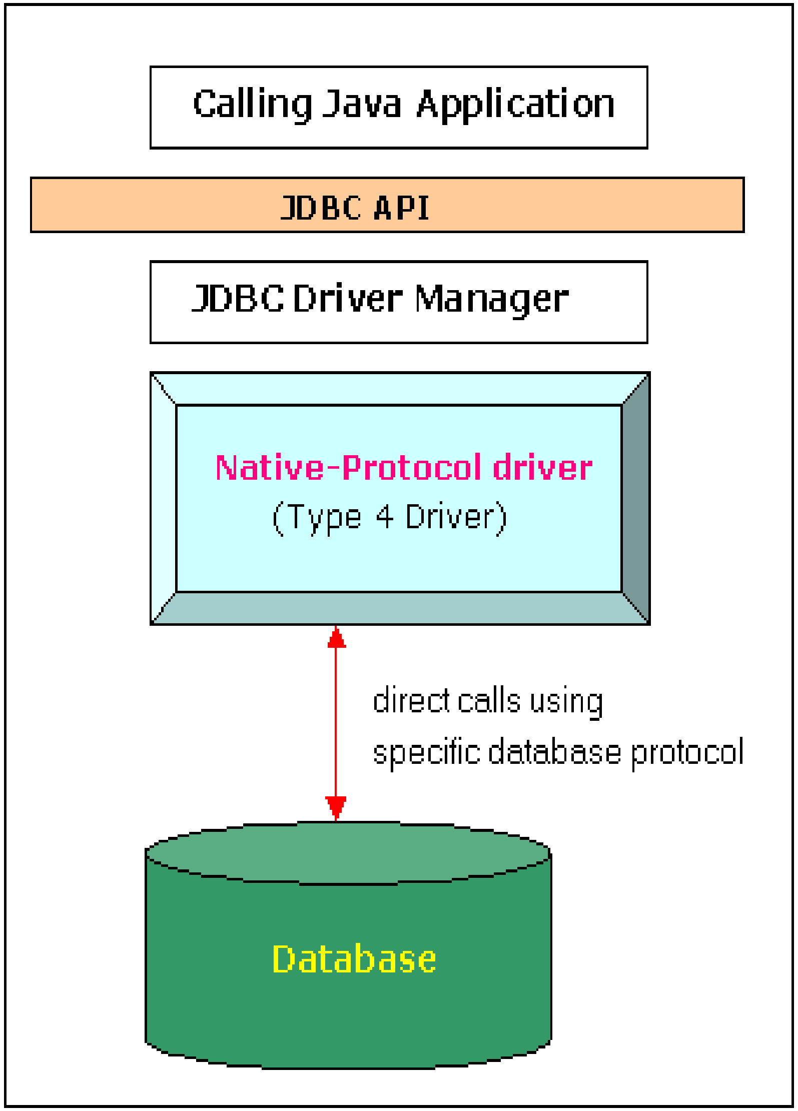
JDBC Driver List

Type IV Drivers

- **100% Pure Java -- the Holy Grail**
- **Communicate directly with a vendor's database through socket connection**
- **Use Java networking libraries to talk directly to database engines**
- **e.g include the widely used Oracle thin driver - oracle.jdbc.driver. OracleDriver**

Go Back To

JDBC Driver List



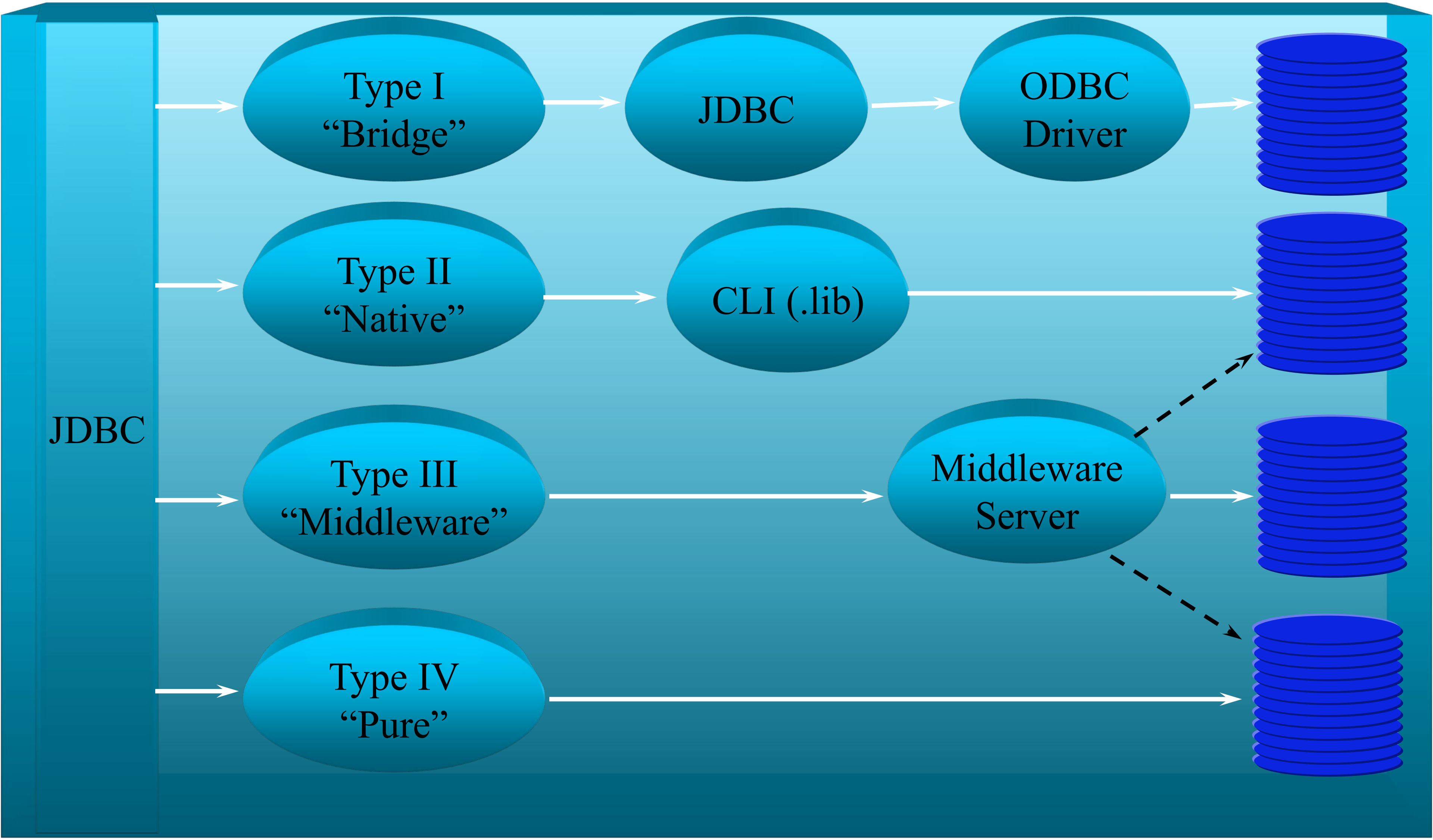
Disadvantage of Type-IV Driver

- **At client side, a separate driver is needed for each database**

Go Back To

JDBC Driver List

JDBC Drivers (Fig.)



Related Technologies

- **ODBC**
  - ✓Requires configuration (odbc.ini)
- **RDO, ADO**
  - ✓Requires Win32
- **JavaBlend**
  - ✓maps objects to tables transparently (more or less)

JDBC API

JDBC API

- The JDBC API classes and interfaces are available in the java.sql and the javax.sql packages.

- The commonly used classes and interfaces in the JDBC API are:

  ✓ **DriverManager class**: Loads the driver for a database.

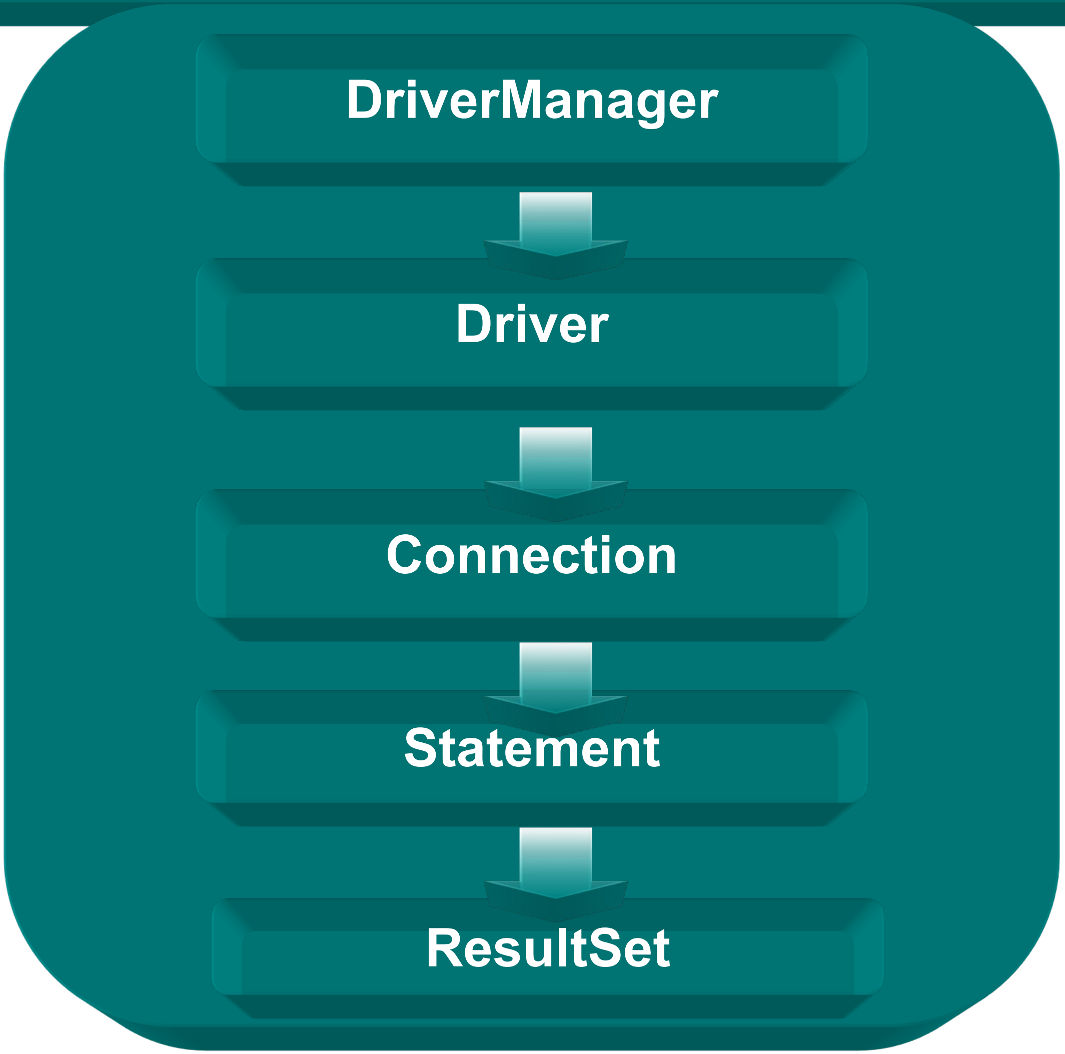
  ✓ **Driver interface**: Represents a database driver. All JDBC driver classes must implement the Driver interface.

  ✓ **Connection interface**: Enables you to establish a connection between a Java application and a database.

JDBC API (Continued)

- ✓ **Statement interface**: Enables you to execute SQL statements.
- ✓ **ResultSet interface**: Represents the information retrieved from a database.
- ✓ **SQLException class**: Provides information about the *exceptions* that occur while interacting with databases.

Steps to create JDBC Application



DriverManager

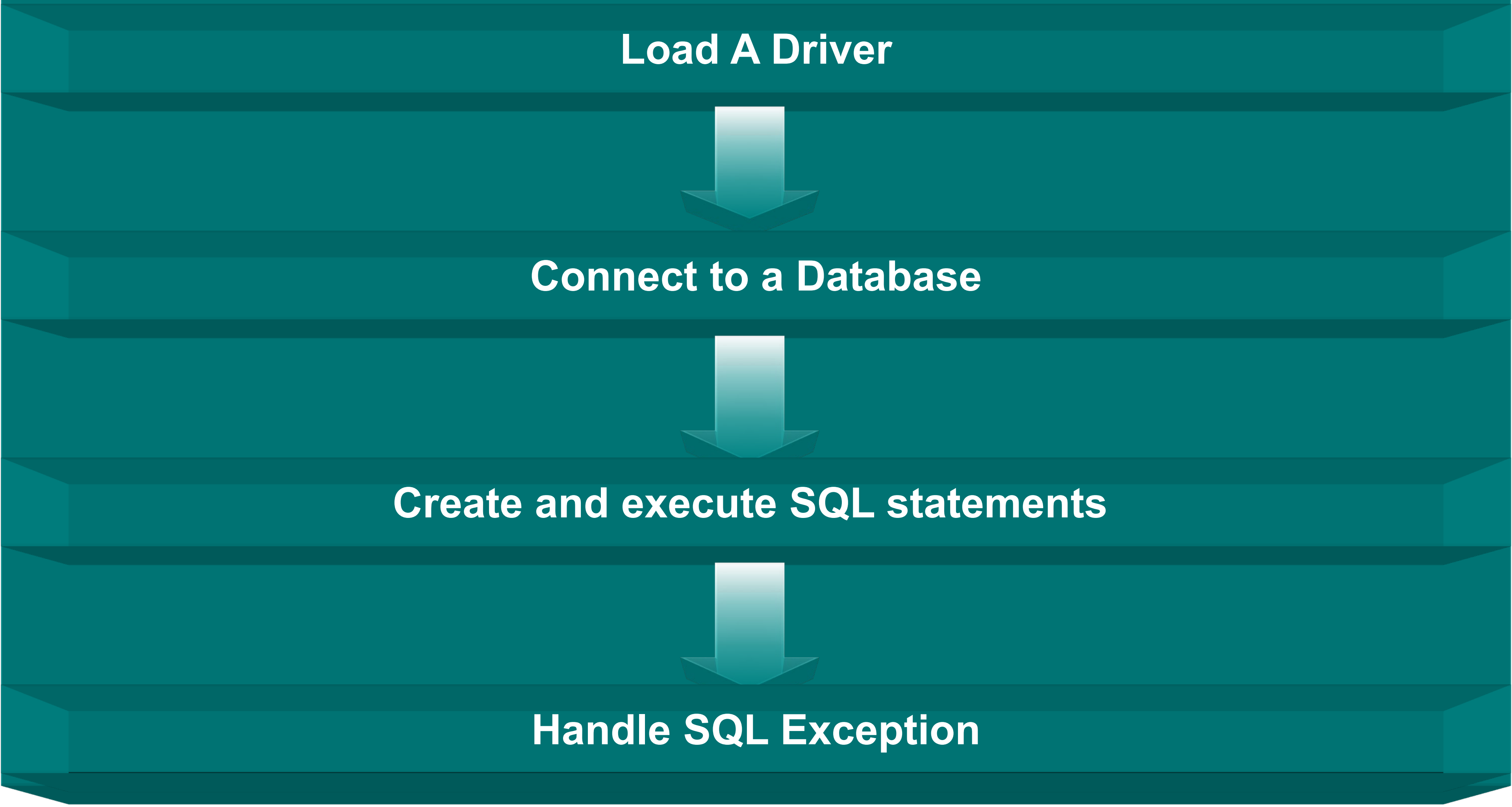
Driver

Connection

Statement

ResultSet

Steps to create JDBC Application (Continued)



Load A Driver

Connect to a Database

Create and execute SQL statements

Handle SQL Exception

JDBC API (Continued)

Load A Driver

- Loading a Driver can be done in two ways:

  - Programmatically:
    - ✓ Using the forName() method
    - ✓ Using the registerDriver()method

  - Manually:
    - ✓ By setting system property

JDBC API (Continued)

Load A Driver (Programmatically)

- Using the forName() method
  - ✓ The forName() method is available in the java.lang.Class class.
  - ✓ The forName() method loads the JDBC driver and registers the driver with the driver manager.
  - ✓ The method call to use the the forName() method is:
  - ✓ Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

JDBC API (Continued)

Load A Driver (Programmatically)

- Using the registerDriver() method
  - ✓ You can create an instance of the Driver class to load a JDBC driver.
  - ✓ This instance provide the name of the driver class at run time.
  - ✓ The statement to create an instance of the Driver class is:

    Driver d = new sun.jdbc.odbc.JdbcOdbcDriver();
  - ✓ You need to call the registerDriver() method to register the Driver object with the DriverManager.
  - ✓ The method call to register the JDBC-ODBC Bridge driver is:

    DriverManager.registerDriver(d);

JDBC API (Continued)

Load A Driver (Manually)

- Setting System Property
  - ✓ To load a JDBC driver, add driver name to the jdbc.drivers system property.
  - ✓ Use the –D command line option to set the system property on the command line.
  - ✓ To set the system property the command is:

```
java –D jdbc.drivers=sun.jdbc.odbc.JdbcOdbcDriver IBMApp
```

JDBC API (Continued)

Connect to a Database

- Connecting to a Database Using DriverManager.getConnection() method:
  - *Connection getConnection (String <url>)*
  - *Connection getConnection (String <url>, String <username>, String <password>)*
    - ✓ Connects to given JDBC URL.
    - ✓ throws java.sql.SQLException
    - ✓ Returns a connection object.

Example:

Connection con=DriverManager.getConnection(“jdbc:odbc:MyDSN”,”scott”,”tiger”);

JDBC API (Continued)

Connect to a Database (Example)

```
String url   = "jdbc:odbc:Northwind";
try {
  Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection con = DriverManager.getConnection(url);
}
catch (ClassNotFoundException e)
  { e.printStackTrace(); }
catch (SQLException e)
  { e.printStackTrace(); }
```

JDBC API (Continued)

Create and Execute SQL Statements

- **The Connection object provides the createStatement() method to create a Statement object.**

```
Statement createStatement()
```

- ✓ returns a new Statement object

```
PreparedStatement prepareStatement(String sql)
```

- ✓ returns a new PreparedStatement object

```
CallableStatement prepareCall(String sql)
```

- ✓ returns a new CallableStatement object

JDBC API (Continued)

Statement Interface

- **A Statement object is used for executing a static SQL statement and obtaining the results produced by it.**

JDBC API (Continued)

Statement Interface Methods

ResultSet executeQuery(String)

- ✓ Execute a SQL statement that returns a single ResultSet.

int executeUpdate(String)

- ✓ Execute a SQL INSERT, UPDATE or DELETE statement. Returns the number of rows changed.

boolean execute(String)

- ✓ Execute a SQL statement that may return multiple results.

JDBC API (Continued)

ResultSet Interface

- **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

JDBC API (Continued)

ResultSet Methods

- **boolean next()**
  - ✓activates the next row
  - ✓the first call to next() activates the first row
  - ✓returns false if there are no more rows
- **void close()**
  - ✓disposes of the ResultSet
  - ✓allows you to re-use the Statement that created it

JDBC API (Continued)

ResultSet Methods (Continued)

- *Type* get*Type*(int columnIndex)
  - ✓returns the given field as the given type
  - ✓fields indexed starting at 1 (not 0)
- *Type* get*Type*(String columnName)
  - ✓same, but uses name of field
  - ✓less efficient
- int findColumn(String columnName)
  - ✓looks up column index given column name

JDBC API (Continued)

ResultSet Methods (Continued)

- String getStri
- boolean getB
- byte getByte(
- short getShor
- int getInt(int c
- long getLong

Explore ResultSet Methods

t columnIndex)
le(int columnIndex)
t columnIndex)
nt columnIndex)
Timestamp(int

JDBC API (Continued)

ResultSet Methods (Continued)

Method Name	Description
1. boolean first()	1. Shifts the control of a result set cursor to the first row of the result set.
2. boolean isFirst()	2. checks whether result set cursor points to the first row or not.
3. boolean beforeFirst()	3. moves the cursor before the first row.
4. boolean isbeforeFirst()	4. Checks whether result set cursor moves before the first row.

JDBC API (Continued)

ResultSet Methods (Continued)

	Method Name		Description
5.	boolean last()	5.	Shifts the control to the last row of result set cursor.
6.	boolean isLast()	6.	checks whether result set cursor points to the last row or not.
7.	boolean afterLast()	7.	moves the cursor after the last row.
8.	boolean isAfterLast()	8.	Checks whether result set cursor moves after the last row.

JDBC API (Continued)

ResultSet Methods (Continued)

Method Name	Description
9. boolean next()	9. Shifts the control to the next row of result set.
10. boolean previous()	10. Shifts the control to the previous row of the result set.
11. boolean absolute(int rowno)	11. Shifts the cursor to the row number that you specify as an argument.
12. boolean relative(int rowno)	12. Shifts the cursor relative to the row number that you specify as an argument.

JDBC API (Continued)

ResultSet Methods (Continued)

Method Name	Description
13. void insertRow()	13. Inserts a row in the current result set.
14. void deleteRow()	14. Deletes a row in the current result set.
15. void updateRow()	15. Updates a row of the current resultset.

JDBC API (Continued)

ResultSet Methods (Continued)

Method Name	Description
16. void updateString(col name, String s)	16. Updates the specified column name with the given string value.
17. void updateInt(col name, int x)	17. Updates the specified column name with the given int value.
18. void updateFloat()	18. Updates the specified column name with the given float value.
19. void cancelRowUpdates()	19. Cancels all of the updates in a row.

JDBC API (Continued)

ResultSet Fields

	Field Name		Description
1.	TYPE_FORWARD_ONLY	1.	The ResultSet object can moves forward only from first to last row.
2.	TYPE_SCROLL_SENSITIVE	2.	Indicates ResultSet is scrollable and it reflects changes in the data made by other user.
3.	TYPE_SCROLL_INSENSITIVE	3.	Indicates ResultSet is scrollable and does not reflect changes in the data made by other user.

JDBC API (Continued)

ResultSet Fields

	Field Name		Description
4.	CONCUR_READ_ONLY	4.	Does not allow to update the ResultSet object.
5.	CONCUR_UPDATABLE	5.	Allows to update the ResultSet object .

JDBC API (Continued)

SQL Syntax

INSERT INTO *table* (*field1, field2*) VALUES (*value1, value2*)

- ✓ inserts a new record into the named table

UPDATE *table* SET (*field1* = *value1, field2* = *value2*) WHERE *condition*

- ✓ changes an existing record or records

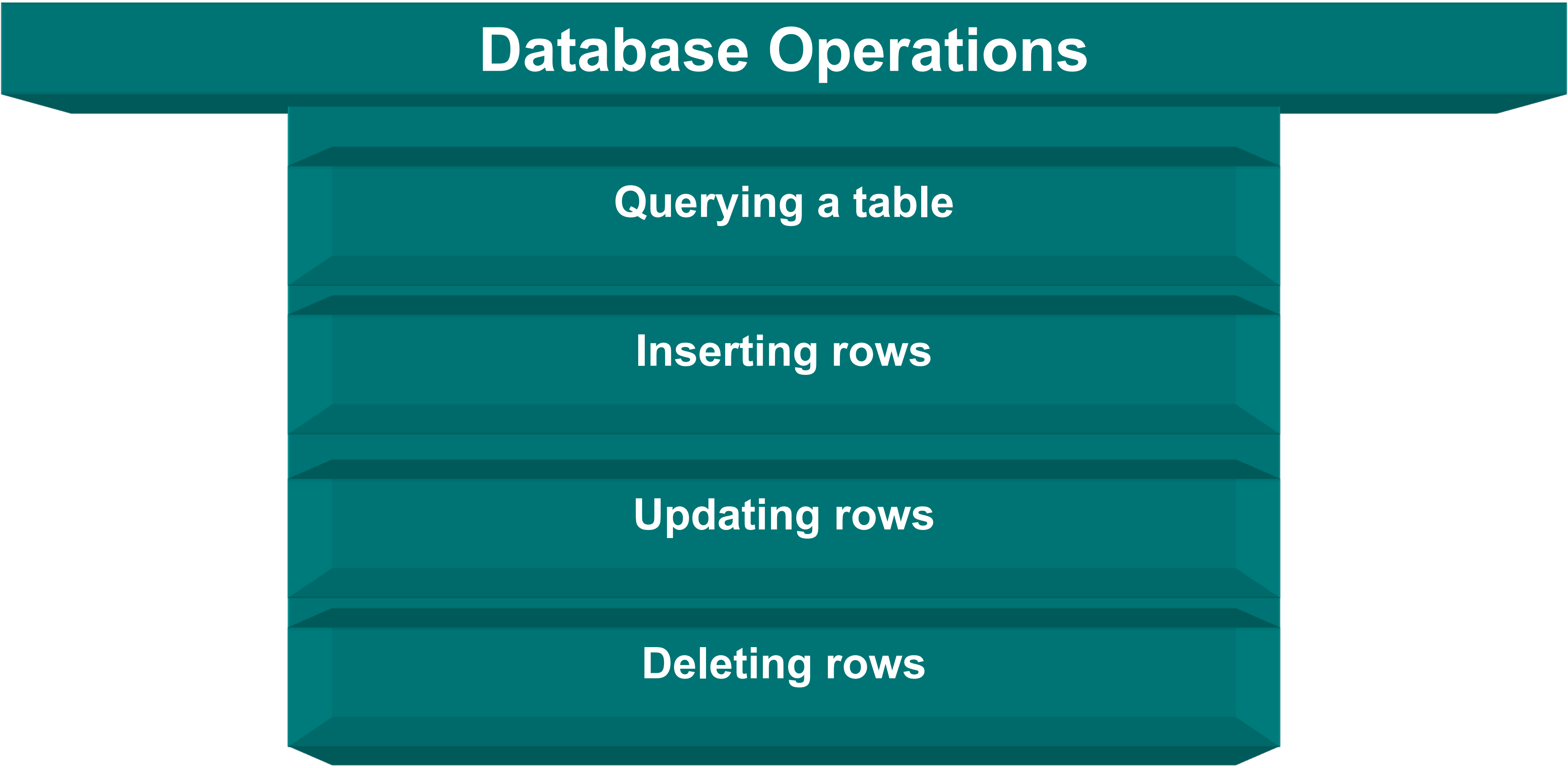
DELETE FROM *table* WHERE *condition*

- ✓ removes all records that match condition

SELECT *field1, field2* FROM *table* WHERE *condition*

- ✓ retrieves all records that match condition

JDBC API (Continued)



Database Operations

Querying a table

Inserting rows

Updating rows

Deleting rows

JDBC API (Continued)

Database Operations

Querying a table

The code snippet to retrieve data from the employees table is:

```
String semp = "SELECT * FROM employees";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(semp);
```

JDBC API (Continued)

Database Operations

Inserting rows

The code snippet to insert rows in employees table is:

```
String semp = “INSERT INTO employees(eid, ename,
basic)  VALUES(1,’A.Sinha’,28000)”;
Statement stmt = con.createStatement();
int noOfInsert = stmt.executeUpdate(semp);
```

JDBC API (Continued)

Database Operations

Updating rows

The code snippet to insert rows in employees table is:

```
String semp = “UPDATE employees SET
basic=basic+2000 where eid=1”;
Statement stmt = con.createStatement();
int noOfUpdate = stmt.executeUpdate(semp);
```

JDBC API (Continued)

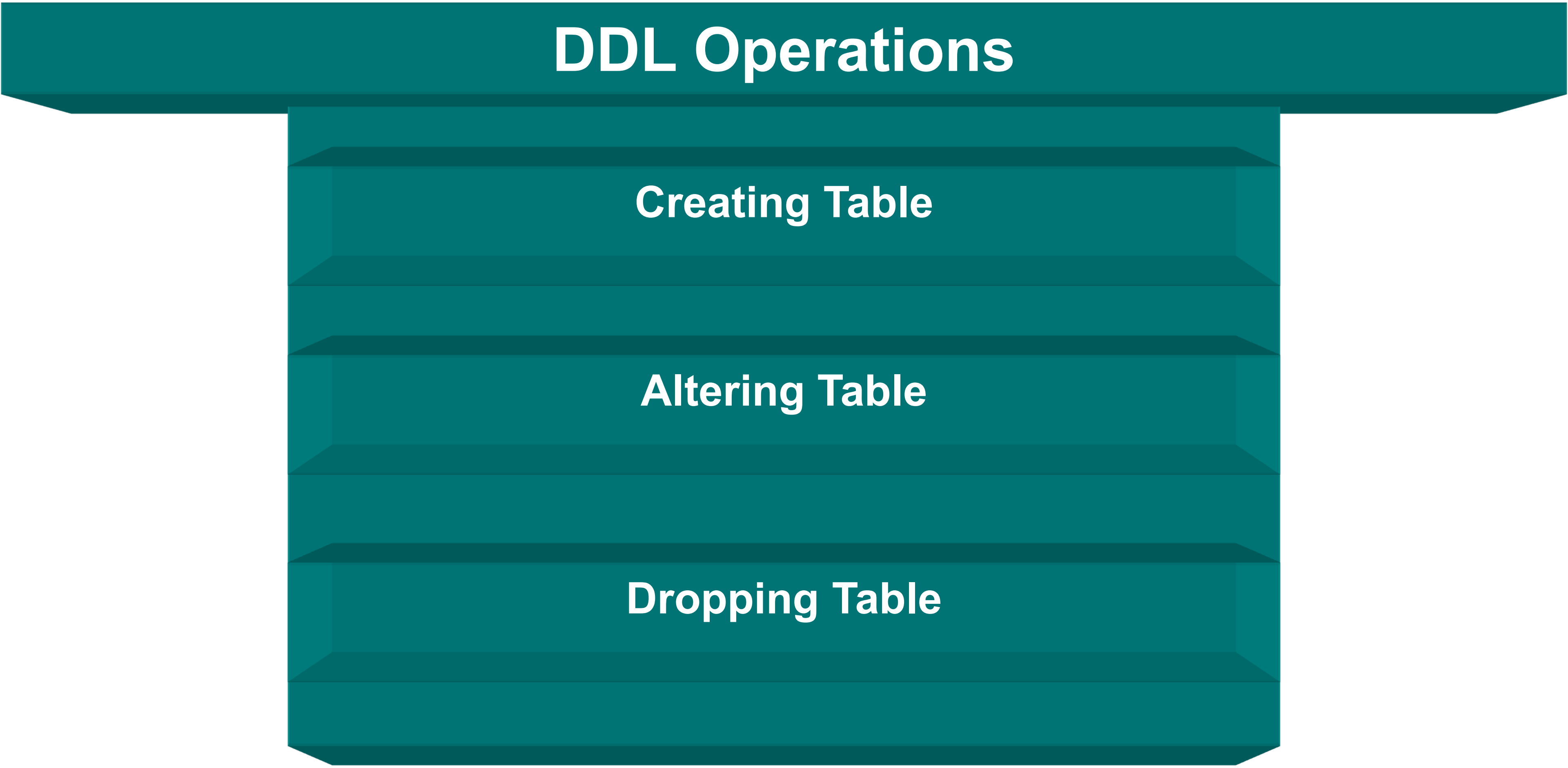
Database Operations

Deleting rows

The code snippet to delete rows in employees table is:

```
String semp = “DELETE FROM employees WHERE
eid=1”;
Statement stmt = con.createStatement();
int noOfDelete = stmt.executeUpdate(semp);
```

JDBC API (Continued)



DDL Operations

Creating Table

Altering Table

Dropping Table

JDBC API (Continued)

DDL Operations

Creating Table

The code snippet to create a *department* table is:

```
Statement stmt = con.createStatement();
stmt.execute(“create table department(eid number(5),
deptno char(10), deptname varchar2(20)” );
```

JDBC API (Continued)

DDL Operations

Altering Table

The code snippet to add a column in *department* table is:

```
Statement stmt = con.createStatement();
stmt.execute(“ALTER TABLE department add depthead
varchar2(15)”);
```

JDBC API (Continued)

DDL Operations

Dropping Table

The code snippet to create a *department* table is:

```
Statement stmt = con.createStatement();
stmt.execute(“DROP TABLE department” );
```

JDBC API (Continued)

PreparedStatement Interface

- The PreparedStatement Interface object:

✓ pass runtime parameters to the SQL statements.

✓ Is compiled and prepared only once by the JDBC.

✓ prepareStatement() method is used to submit parameterized query using a connection object to the database.

JDBC API (Continued)

PreparedStatement Interface (Continued)

Code snippet for preparedStatement:

The code snippet to pass the employee id during runtime using prepareStatement() method:

```
String s=‟select * from employee where eid=? ‟
PreparedStatement pst = con.prepareStatement(s);
pst.setInt(1,100 );
ResultSet rs=pst.executeQuery();
```

It acts as a placeholder

The value of each ‘?’ is set by calling appropriate setXXX() method, In this case setInt()

JDBC API (Continued)

Mapping Java Types to SQL Types

SQL type	Java Type
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Transactions

Transactions Overview

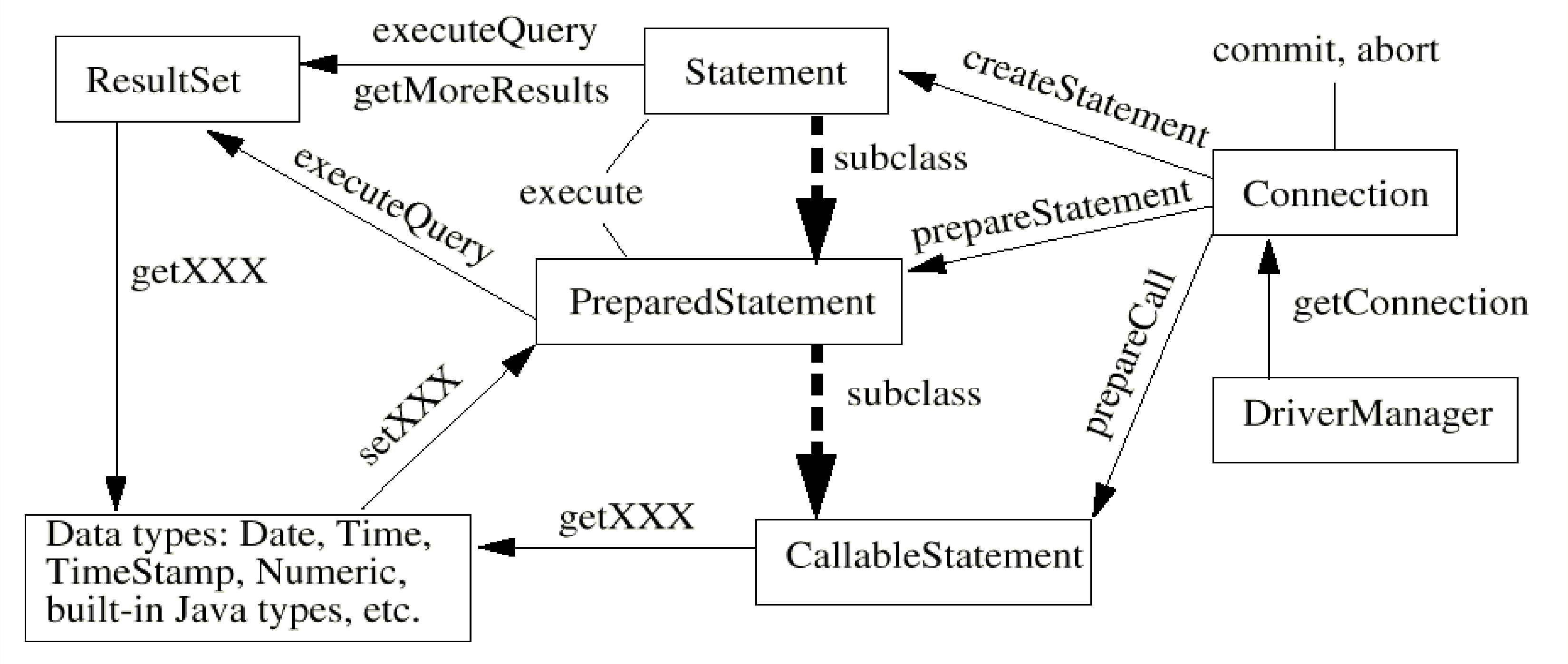
- Transaction = more than one statement which must all succeed (or all fail) together
- If one fails, the system must reverse all previous actions
- Also can’t leave DB in inconsistent state halfway through a transaction
- COMMIT = complete transaction
- ROLLBACK = abort

Transactions (Continued)

Transaction Management

- Transactions are not explicitly opened and closed
- if AutoCommit is true, then every statement is automatically committed
- default case: true
- if *AutoCommit* is false, then every statement is added to an ongoing transaction
- Must explicitly rollback or commit.

JDBC Class Diagram



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

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

```
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:xe
    ","system","oracle");
con.setAutoCommit(false);
 Statement stmt=con.createStatement();
stmt.addBatch("insert into user420 values(190,‘palak',40000)");
stmt.addBatch("insert into user420 values(191,‘prakriti',50000)");
stmt.executeBatch();//executing the batch


con.commit();
con.close();
}}
```

Summary

- JDBC Architecture consists of two layers:
  - ✓ JDBC application layer: Signifies a Java application that uses the JDBC API to interact with the JDBC driver manager.
  - ✓ JDBC driver layer: Contains a driver, such as an SQL Server driver, which enables a Java application to connect to a database. This layer acts as an interface between a Java application and a database.
- The JDBC driver manager manages various JDBC drivers.
- The JDBC driver is software that a Java application uses to access a database.

Summary (Continued)

- JDBC supports four types of drivers:
  - ✓ JDBC-ODBC Bridge driver
  - ✓ Native-API Partly-Java driver
  - ✓ JDBC-Net Pure-Java driver
  - ✓ Native Protocol Pure-Java driver
- The JDBC API consists of various classes and interfaces that enable Java applications to interact with databases.
- The classes and interfaces of the JDBC API are defined in the java.sql and javax.sql packages.
- You can load a driver and register it with the driver manager either programmatically or manually.

Summary (Continued)

- Two ways to load and register a driver programmatically are:
  - ✓ Using the Class.forName() method
  - ✓ Using the registerDriver() method
- You can add the driver name to the jdbc.drivers system property to load and register a JDBC driver manually.
- A Connection object establishes a connection between a Java application and a database.
- A Statement object sends requests to and retrieves results from a database.
- You can insert, update, and delete data from a table using the DML statements in Java applications.

Summary (Continued)

- You can create, alter, and drop tables from a database using the DDL statements in Java applications.
- A ResultSet object stores the result retrieved from a database when a SELECT statement is executed.
- You can create various types of ResultSet objects such as read only, updatable, and forward only.

Test Your Understanding

1. A JDBC ___________ is a software that a Java application uses to access a database.

   [✓ a. Driver]

   b. DSN

   c. CLI

   d. DriverManager

2. _ _ _ _ _ _ interface enables you to establish a connection between a Java application and the database.

   [✓ a. ResultSet]

   b. Connection

   c. Statement

   d. SQL

Test Your Understanding (Contd.)

3. _ _ _ _ _ manages various JDBC drivers?

   a. ManagerDriver

   b. Driver

   [✓ c. DriverManager]

   d. ODBC Driver

4. _ _ _ _ _ _ _sends requests to and retrieves result from a database.

   a. ResultSet

   [✓ b. Statement]

   c. Connection

   d. next()