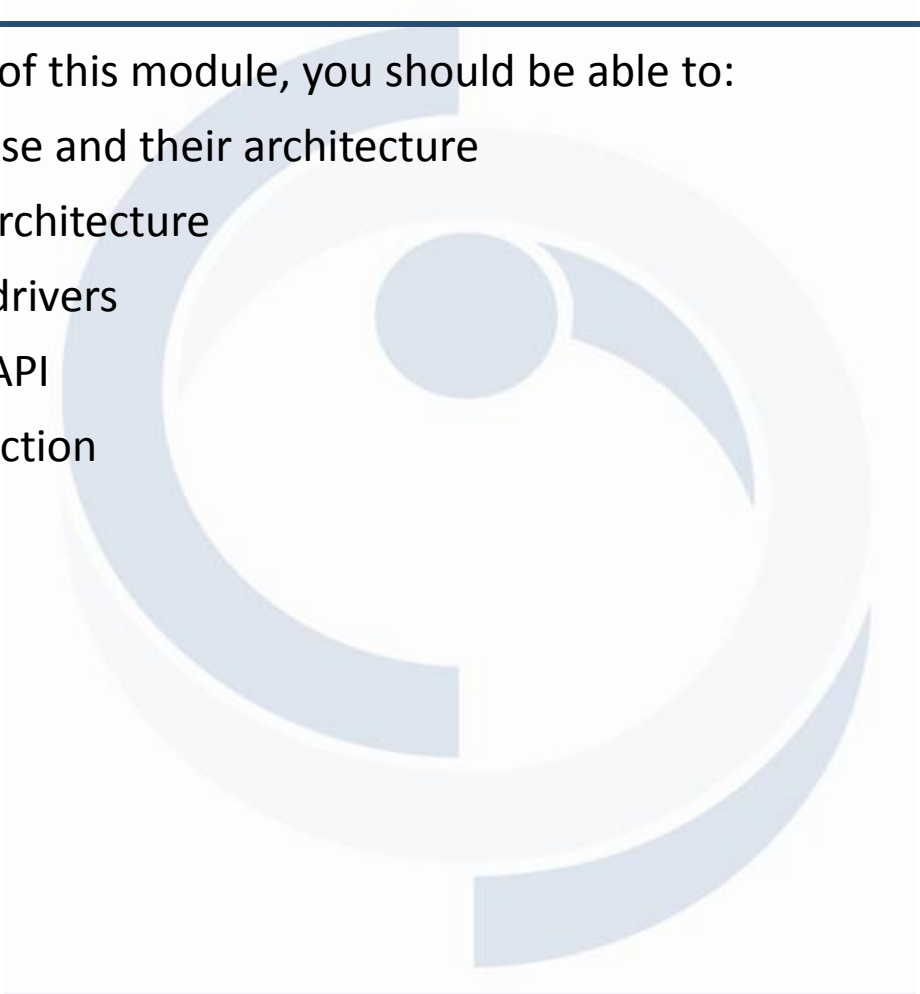
Core Java

Java DataBase Connectivity

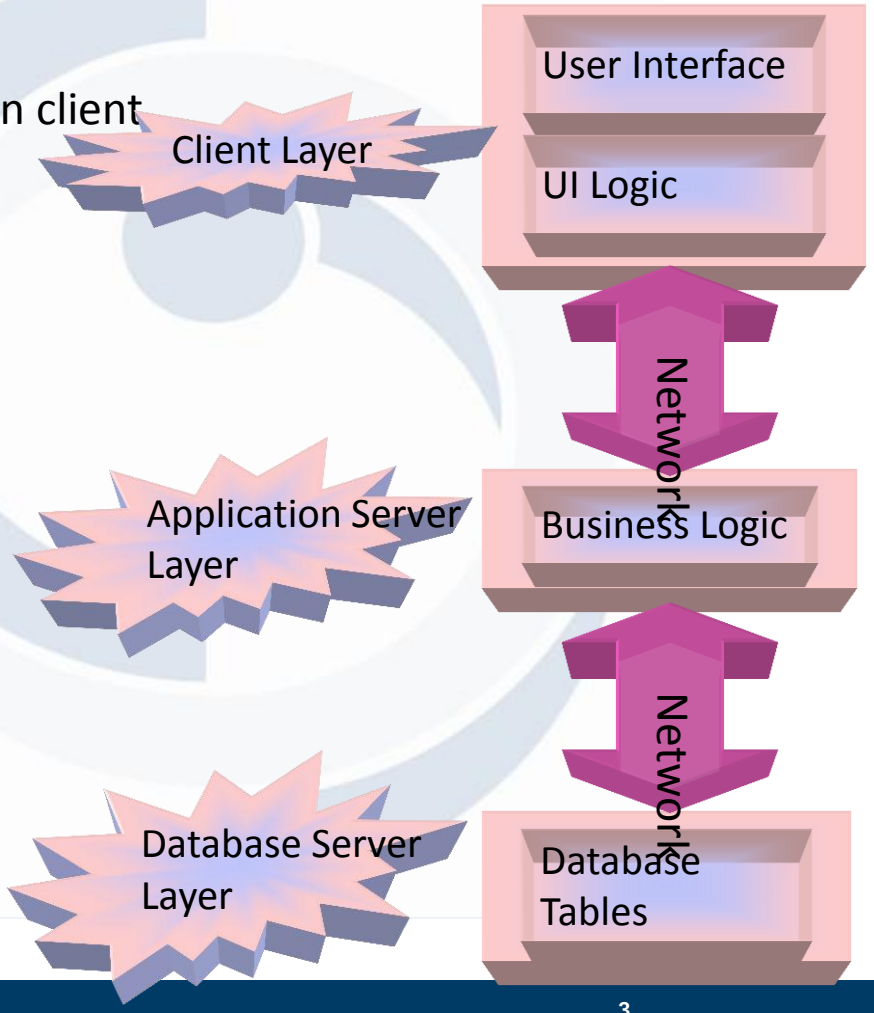
After completion of this module, you should be able to:

- Define database and their architecture
- Define JDBC architecture
- Explain JDBC drivers
- Explain JDBC API
- Explain transaction



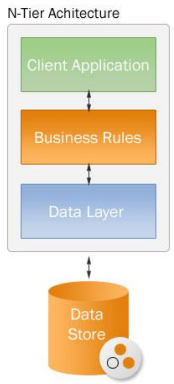
Three-Tier Architecture

- Application Server sits between client and database.



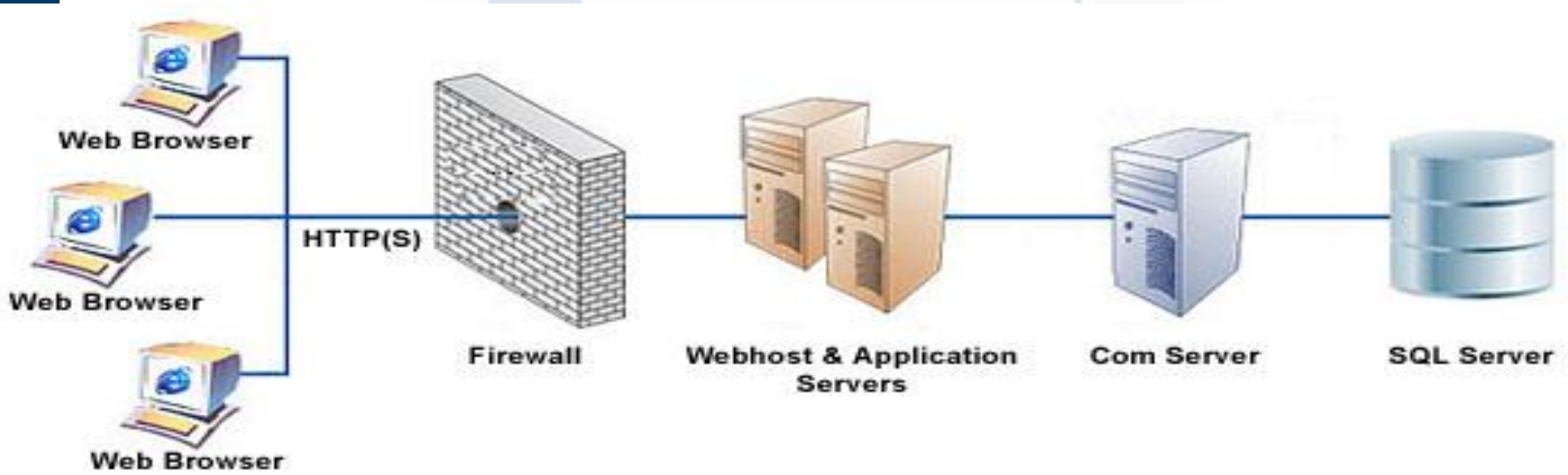
Three-tier pros

- 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



N-tier architecture

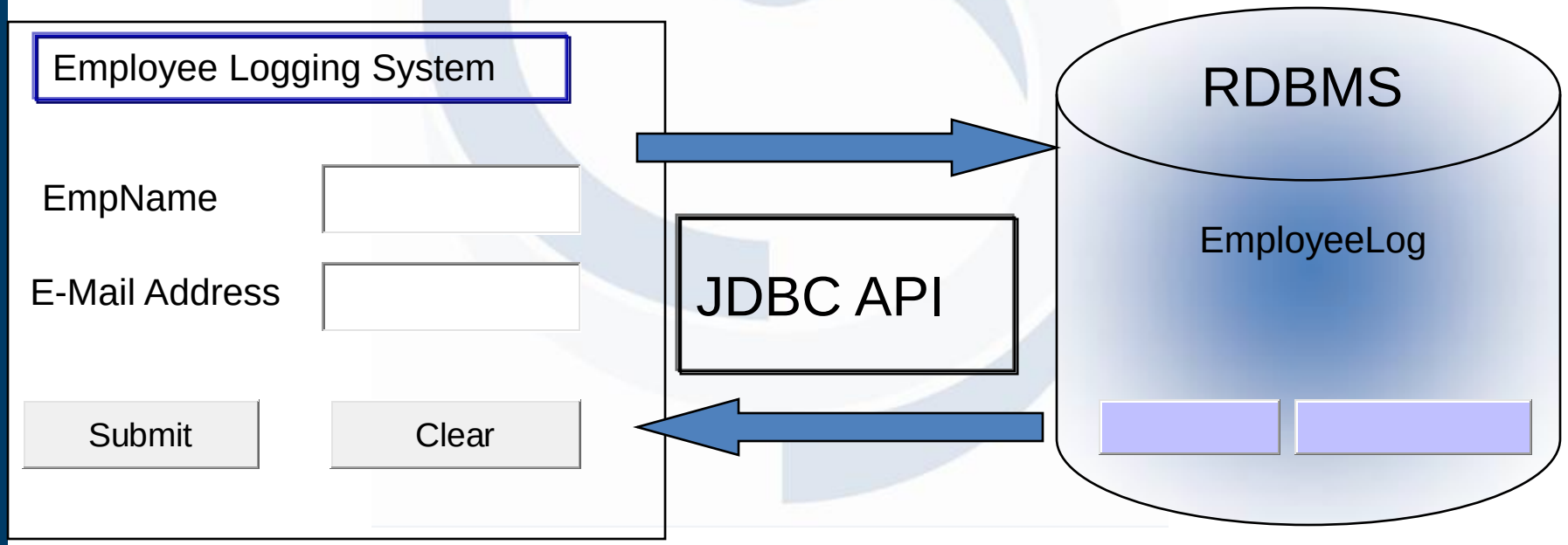
- **Design your application using as many “tiers” as you need**
- **Use Object-Oriented Design techniques**
- **Put the various components on whatever host makes sense**
- **Java allows N-Tier Architecture, especially with RMI and JDBC**



Java application connects to database

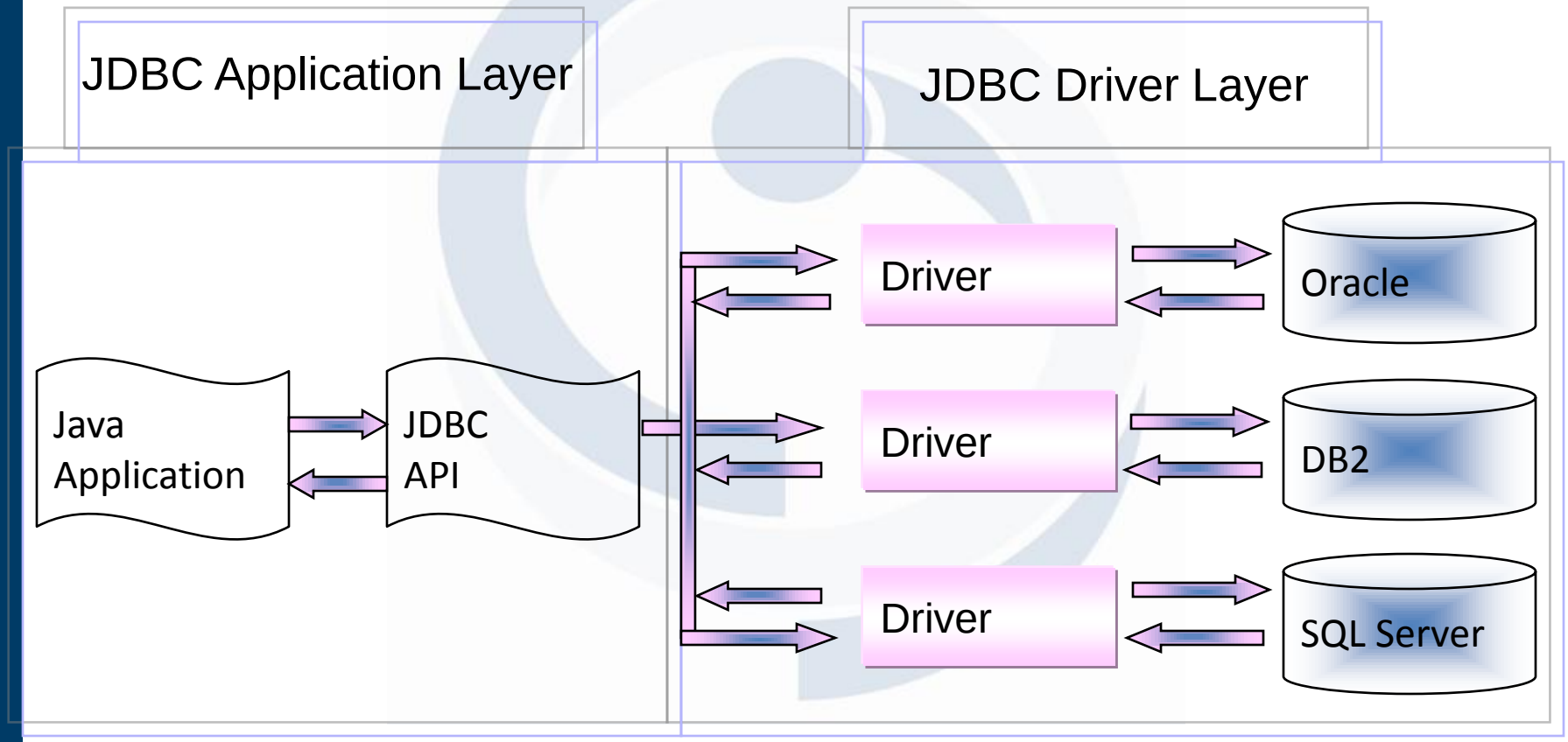
Java Application  Database

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

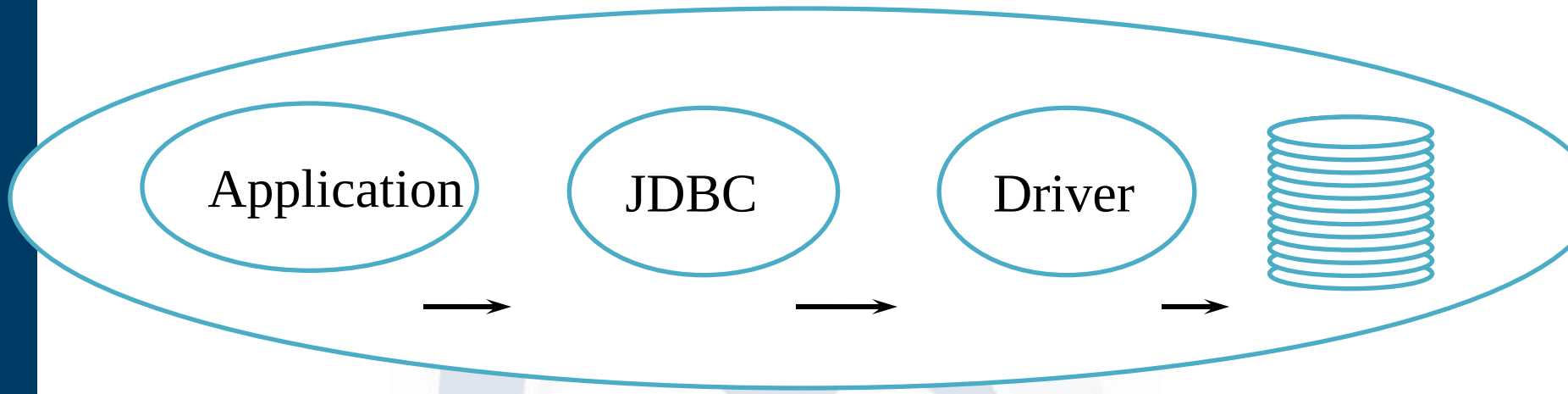


JDBC architecture

- It can be categorized into into two layers:



JDBC architecture (continued)



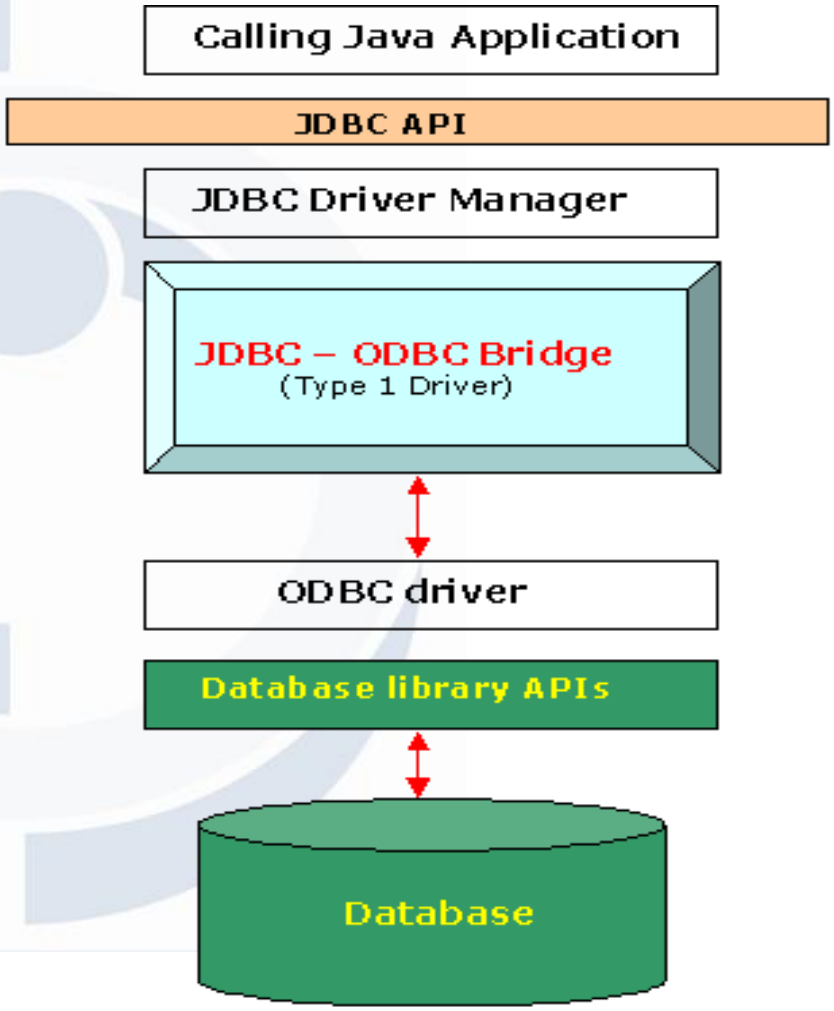
- **Java code calls JDBC library**
- **JDBC loads a *driver***
- **Driver talks to a particular database**
- **Can have more than one driver -> more than one database**
- **Can change database engines without changing any application code**

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

Type 1 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.

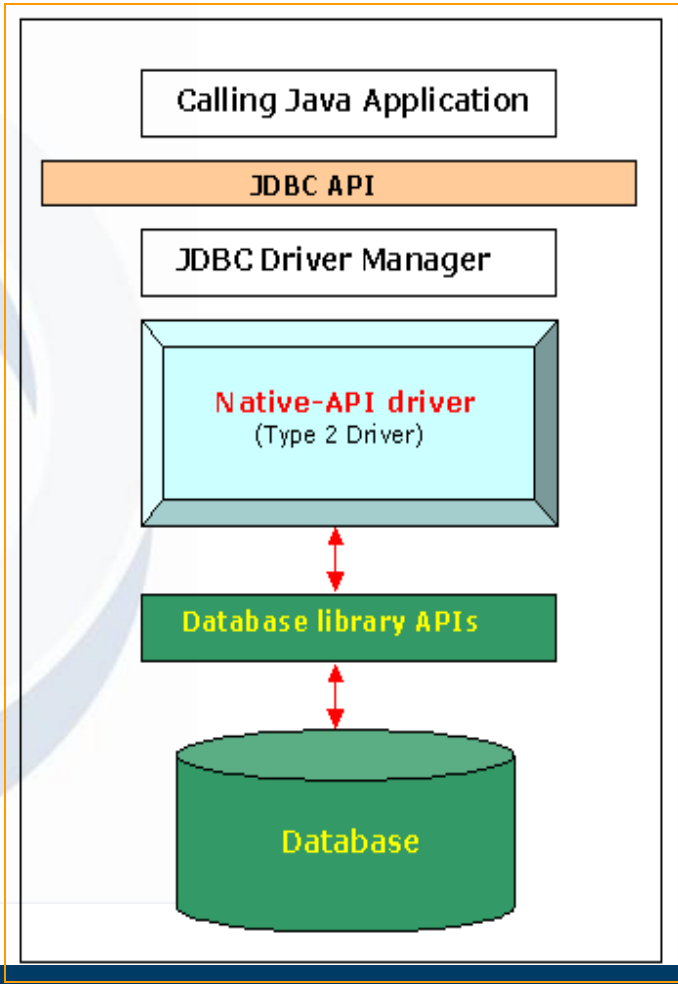


Disadvantage of Type-I Driver

- **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**

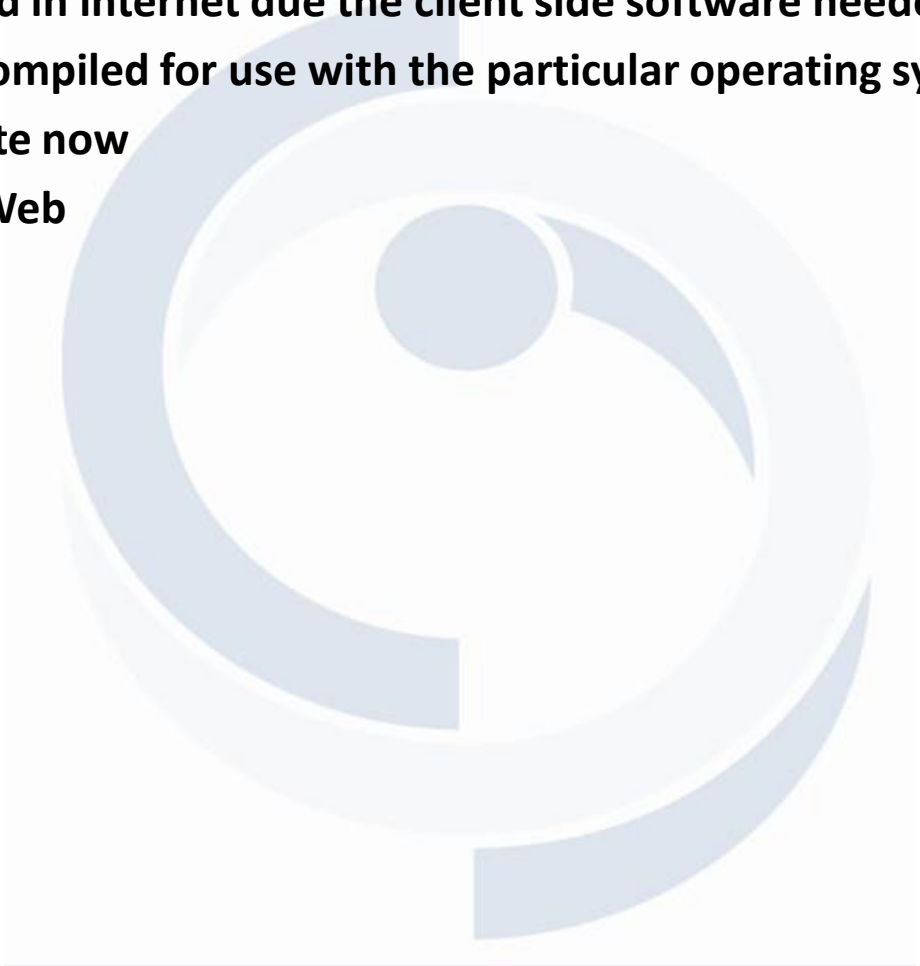
Type II drivers

- **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.**



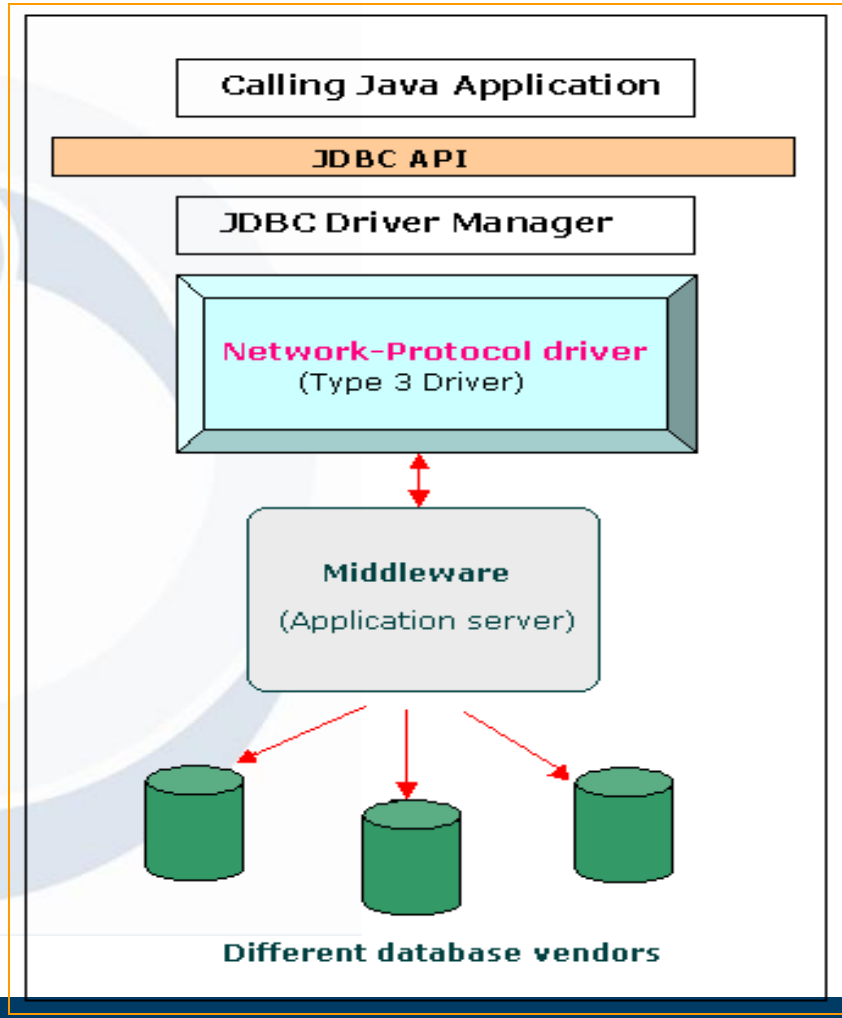
Disadvantage of Type-II Driver

- **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**



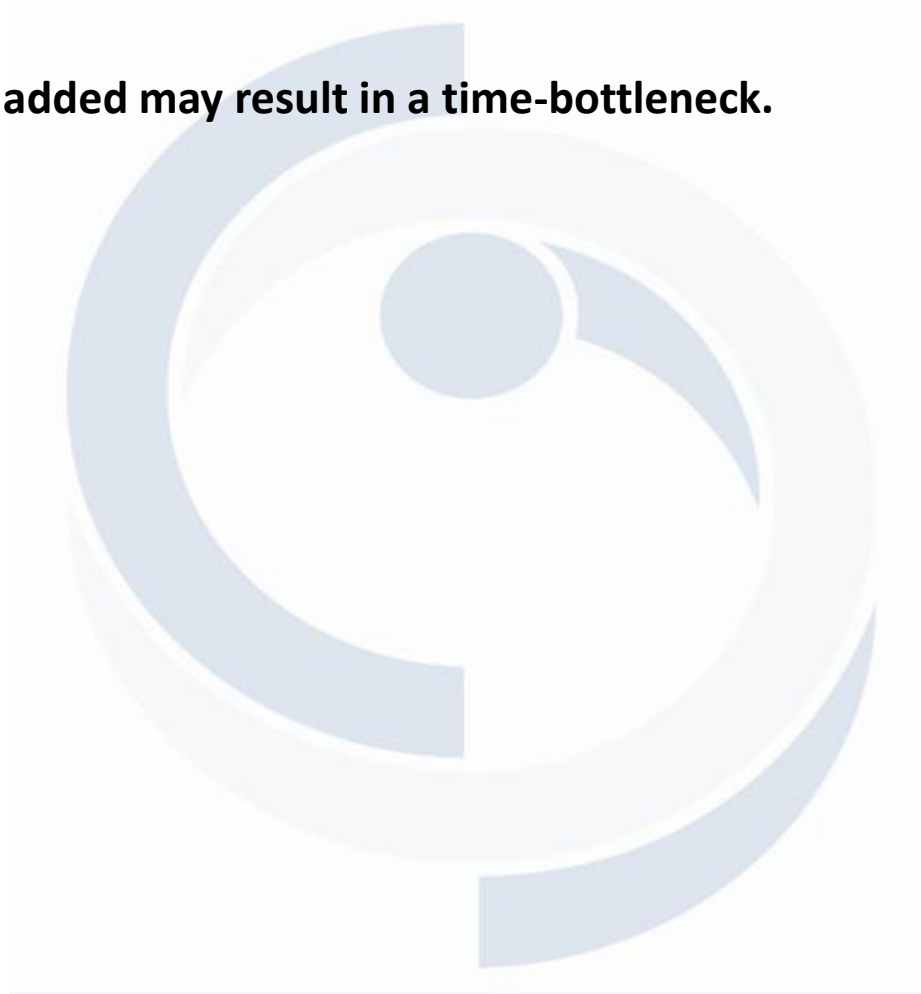
Type III drivers

- **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**



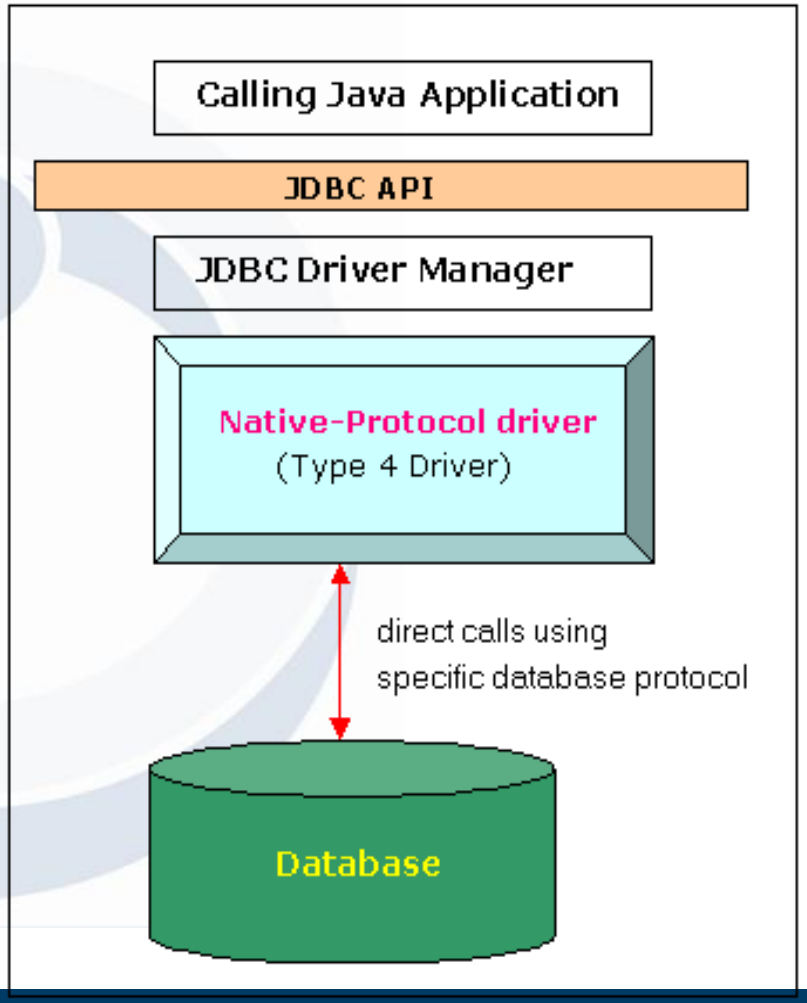
Disadvantage of Type-III Driver

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



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**



Disadvantage of Type-IV Driver

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

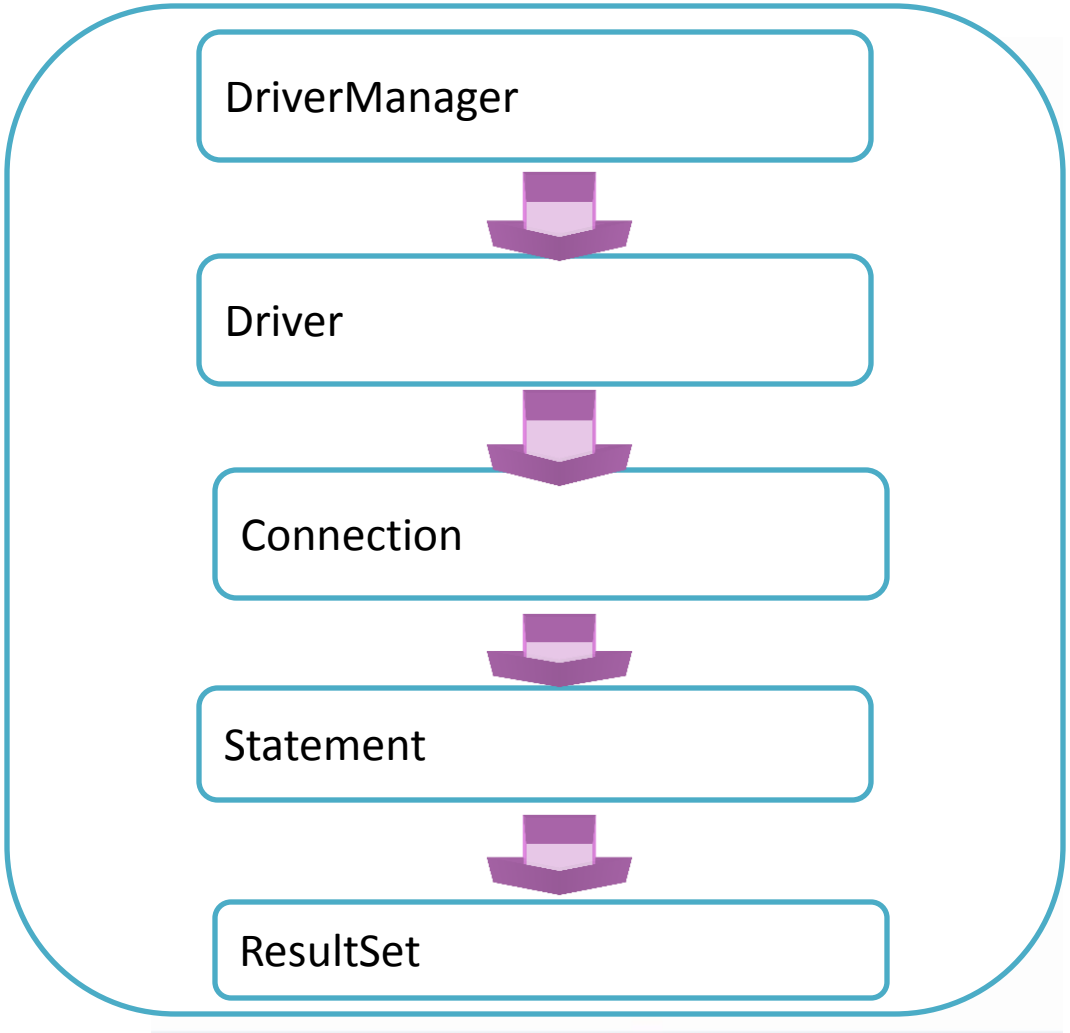
JDBC API

- 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



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)

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

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

  Statement createStatement()
  - returns a new Statement object.
- **Prepared Statement Interface**

  When you use a PreparedStatement object to execute a SQL statement, the statement is parsed and compiled by the database, and then placed in a statement cache. From then on, each time you execute the same PreparedStatement, it is once again parsed, but no recompile occurs. Instead, the precompiled statement is found in the cache and is reused.

  PreparedStatement prepareStatement(String sql)
  - returns a new PreparedStatement object.

JDBC API (continued)

Create and Execute SQL Statements

- **CallableStatement Interface**
  - The interface used to execute SQL stored procedures.
  - Syntax:
    - CallableStatement cl = con.prepareCall("{call show_emp}");
      - Where, **con** is the connection identifier and **show_emp** is the procedure name.

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 getType(int columnIndex)**
  - returns the given field as the given type
  - fields indexed starting at 1 (not 0)
- **Type getType(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)

1. **String getString(int columnIndex)**
2. **boolean getBoolean(int columnIndex)**
3. **byte getByte(int columnIndex)**
4. **short getShort(int columnIndex)**
5. **int getInt(int columnIndex)**
6. **long getLong(int columnIndex)**

1. float getFloat(int columnIndex)
2. double getDouble(int columnIndex)
3. Date getDate(int columnIndex)
4. Time getTime(int columnIndex)
5. Timestamp getTimestamp(int columnIndex)

JDBC API (continued)

ResultSet Methods (Continued)

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)

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)

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 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	Does not allow to update the ResultSet object.
5.	CONCUR_UPDATABLE	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 d*epartment* 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();
```

JDBC API (continued)

CallableStatement Interface

Code snippet for Creating show_emp Stored Procedure:

```
String crProc="create procedure show_emp as
select  *  from  emp";
Statement st = con.createStatement();
st.executeUpdate(crProc);
```

JDBC API (continued)

CallableStatement Interface

Code snippet for Calling show_emp Stored Procedure:

```
CallableStatement clst = con.prepareCall("{call
show_emp}");
ResultSet rs = clst.executeQuery();
// ... Fetch the Data
```

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

Transaction

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**
- **Syntax**
- **Connection con.setAutoCommit(true); //statements are committed automatically**
- **Connection con.setAutoCommit(false);// statements are not committed automatically**

Transaction (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.**

Exercise

- **Create the following Menu as shown below:**

********** **Book Details**
****

1. **Create A Book Table**
2. **Drop The Book Table**
3. **Alter The Book Table**
4. **Insert A New Book**
5. **Delete A Book**
6. **Update A Book**
7. **Retrieve Book Details**
8. **Exit**

****
****

Exercise (continued)

- **Description of the Menu Option's**

1. **Create A Book Table – The structure of the table is given below:**

Field Name	**DataType**
BOOK_ID	NUMBER(4)
BOOK_NAME	VARCHAR2(20)
PUB_NAME	CHAR(15)
PRICE	NUMBER(8,2)

2. **Drop the Book Table created in the 1st option.**
3. **Allow to alter the structure of the Book Table by adding a new column named Description with VARCHAR2(30) as type.**