Object Oriented Development with Java

(CT038-3-2 and Version VC1)

Java Connectivity Database (JDBC)

Topic & Structure of The Lesson

- Introduction to JDBC
- JDBC Architecture
- Seven Steps in JDBC connection

Learning Outcomes

- **At the end of this topic, You should be able to**

  -describe the physical structure of JDBC Architecture

  - describe the connection between the application and the database

Key terms you must be able to use

If you have mastered this topic, you should be able to use the following terms correctly in your assessments:

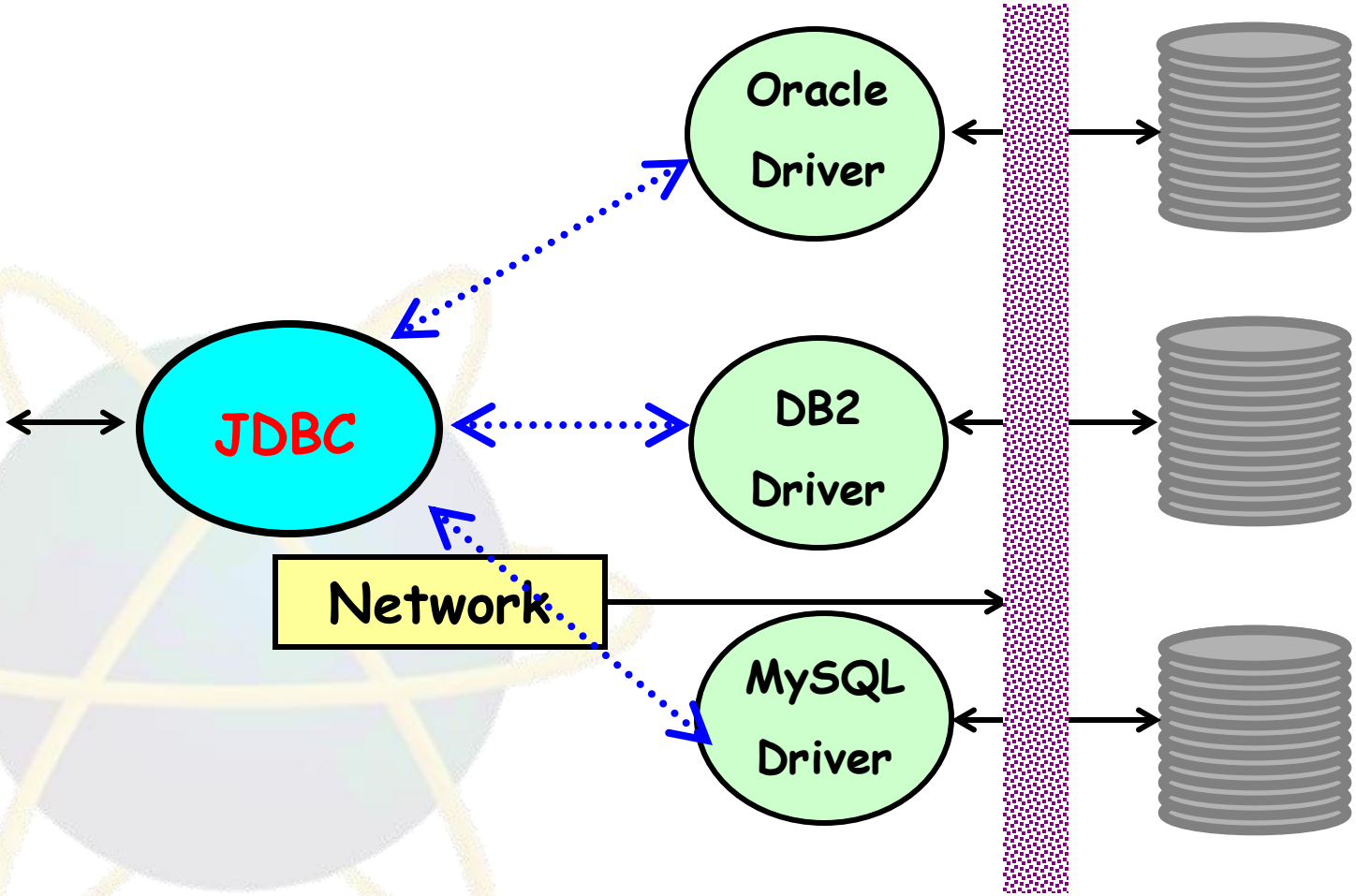
-JDBC

- Database connectivity

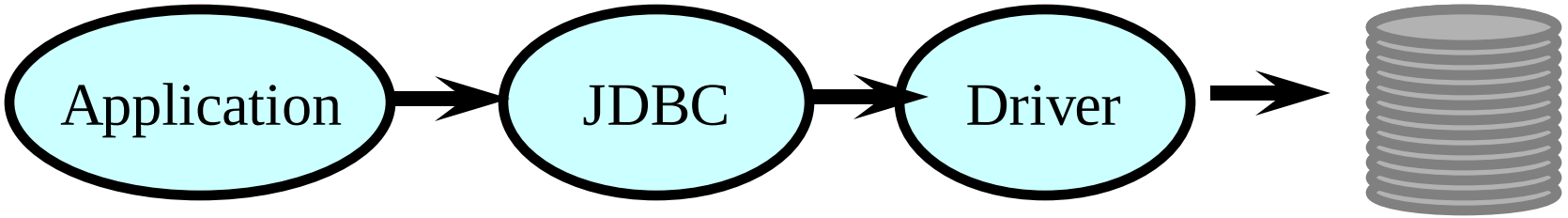
Introduction to JDBC

- **JDBC** is used for accessing databases from Java applications
- Information is transferred from relations to objects and vice-versa
  - *databases* optimized for *searching/indexing*
  - *objects* optimized for *engineering/flexibility*

JDBC Architecture



JDBC Architecture (cont.)



-Java code calls JDBC library

-JDBC loads a *driver*

-Driver talks to a particular database

-An application can work with several databases by using all corresponding drivers

Ideal: can change database engines *without changing any application code* (not always in practice)

Seven Steps

- Load the driver
- Define the connection URL
- Establish the connection
- Create a Statement object
- Execute a query using the Statement
- Process the result
- Close the connection

Loading the Driver

- We can register the driver indirectly using the statement

```
Class.forName("com.mysql.jdbc.Driver");
```

- Class.forName loads the specified class
- When mysqlDriver is loaded, it automatically
  - creates an instance of itself
  - registers this instance with the DriverManager
- Hence, the driver class can be given as an argument of the application

Connecting to the Database

A·P·U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

- Every database is identified by a URL
- Given a URL, DriverManager looks for the driver that can talk to the corresponding database
- DriverManager tries all registered drivers, until a suitable one is found

Interaction with the Database

- We use Statement objects in order to
  - Query the database
  - Update the database
- Three different interfaces are used:
  Statement, PreparedStatement, CallableStatement
- All are interfaces, hence cannot be instantiated
- They are created by the Connection

Querying with Statement

- String querySt r =

```
"SELECT * FROM employee " +"WHERE lname = 'Wong'";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(queryStr);
```

The executeQuery method returns a ResultSet object representing the query result.

Changing DB with Statement

- String deleteStr =

```
"DELETE FROM employee " +"WHERE lname = 'Wong'";
 Statement stmt = con.createStatement();
 int delnum = stmt.executeUpdate(deleteStr);
```

- executeUpdate is used for data manipulation: insert, delete, update, create table, etc. (anything other than querying!)
- executeUpdate returns the number of rows modified

About Prepared Statements

- Prepared Statements are used for queries that are executed many times
- They are parsed (compiled) by the DBMS only once
- Column values can be set after compilation
- Instead of values, use '?'
- Hence, Prepared Statements can be though of as statements that contain placeholders to be substituted later with actual values

Querying with PreparedStatement

- String queryStr = "SELECT * FROM employee "
  +"WHERE superssn= ? and salary > ?";

```
PreparedStatement pstmt =
    con.prepareStatement(queryStr);
pstmt.setString(1, "333445555");
pstmt.setInt(2, 26000);

ResultSet rs = pstmt.executeQuery();
```

ResultSet

- ResultSet objects provide access to the tables generated as results of executing a Statement queries
- Only one ResultSet per Statement can be open at the same time!
- The table rows are retrieved in sequence
  - A ResultSet maintains a cursor pointing to its current row
  - The next() method moves the cursor to the next row

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
  - automatically called by most Statement methods

ResultSet Methods

- *Type* get*Type*(int columnIndex)
  - returns the given field as the given type
  - indices start at 1 and not 0!
- *Type* get*Type*(String columnName)
  - same, but uses name of field
  - less efficient
- For example: getString(columnIndex), getInt(columnName), getTime, getBoolean, getType,...
- int findColumn(String columnName)
  - looks up column index given column name

ResultSet Methods

JDBC 2.0 includes scrollable result sets. Additional methods included are : ‘first’, ‘last’, ‘previous’, and other methods.

ResultSet Example

- Statement stmt = con.createStatement();

```
ResultSet rs = stmt.executeQuery("select lname,salary fro
m Employees");
// Print the result
    while(rs.next()) {
       System.out.print(rs.getString(1) + ":");
       System.out.println(rs.getDouble("salary"));
        }
```

Quick Review Question

- What is JDBC
- Structure of JDBC
- What are different Steps involved in Java Database Connectivity

Summary of Main Teaching Points

- JDBC Architecture
- Seven Steps in JDBC connection

Question and Answer Session

A·P·U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

Q & A

Next Session

- Collections
- Collection Interface
- Set Interface
- Hash set
- Linked Hash set
- Tree set
- List
- ArrayList
- -Vector