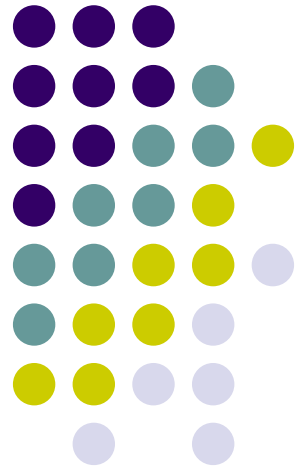
JDBC Tutorial

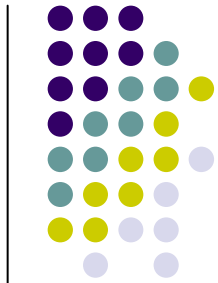
MIE456 -
Information Systems Infrastructure II

Vinod Muthusamy

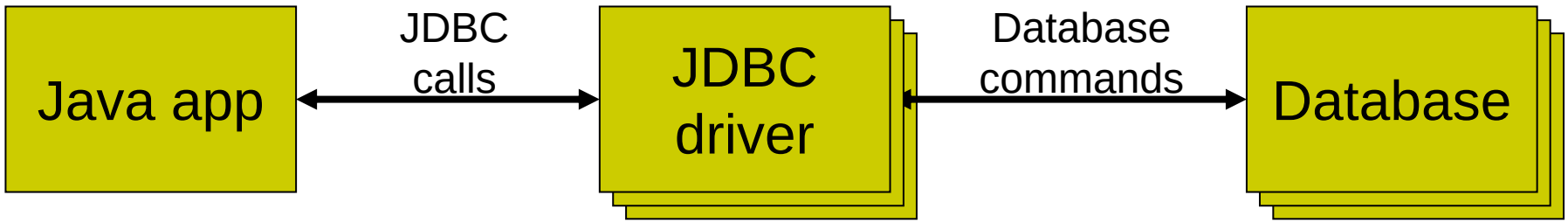
November 4, 2004



Java Database Connectivity (JDBC)

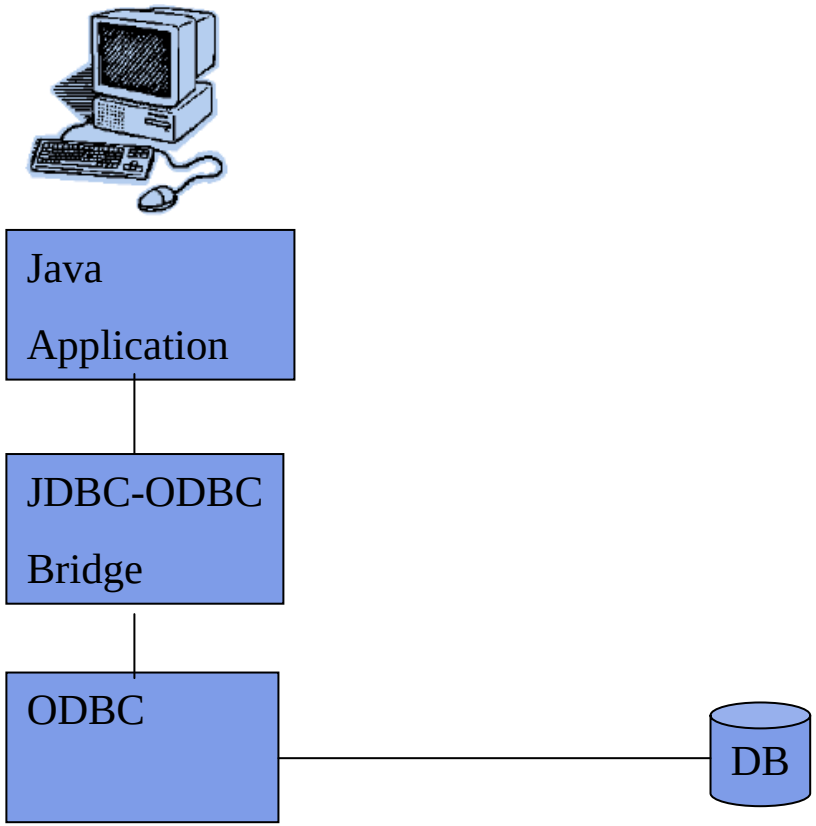


- An interface to communicate with a relational database
  - Allows database agnostic Java code
  - Treat database tables/rows/columns as Java objects
- JDBC driver
  - An implementation of the JDBC interface
  - Communicates with a particular database



JDBC Driver Types

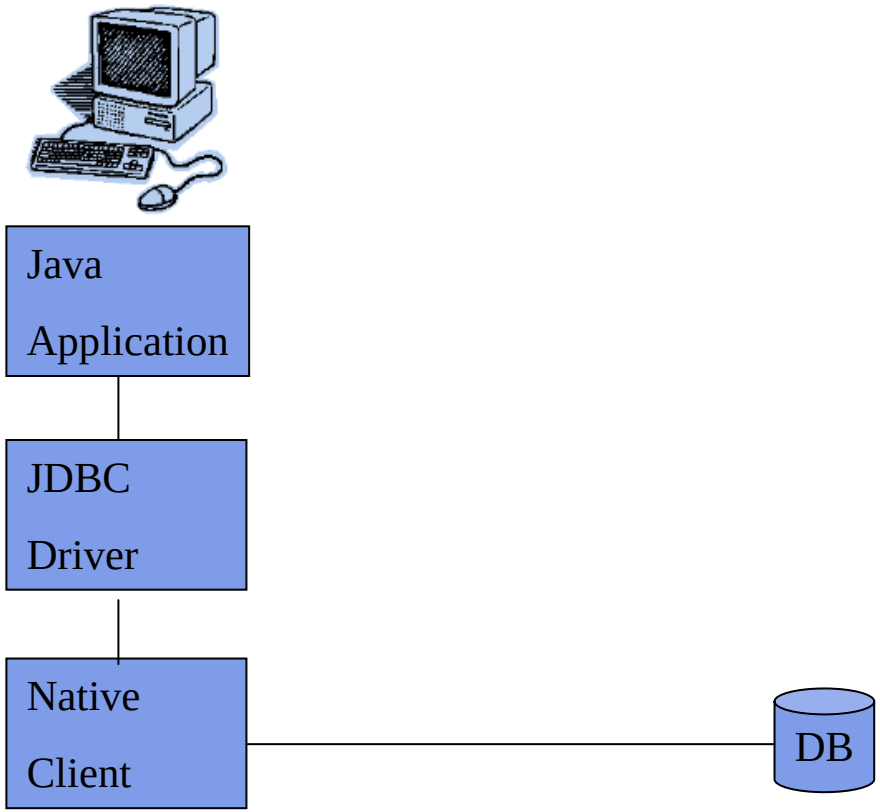
- **Type 1: JDBC-ODBC Bridge**



Note: in following illustrations, DB may be on either a local or remote machine

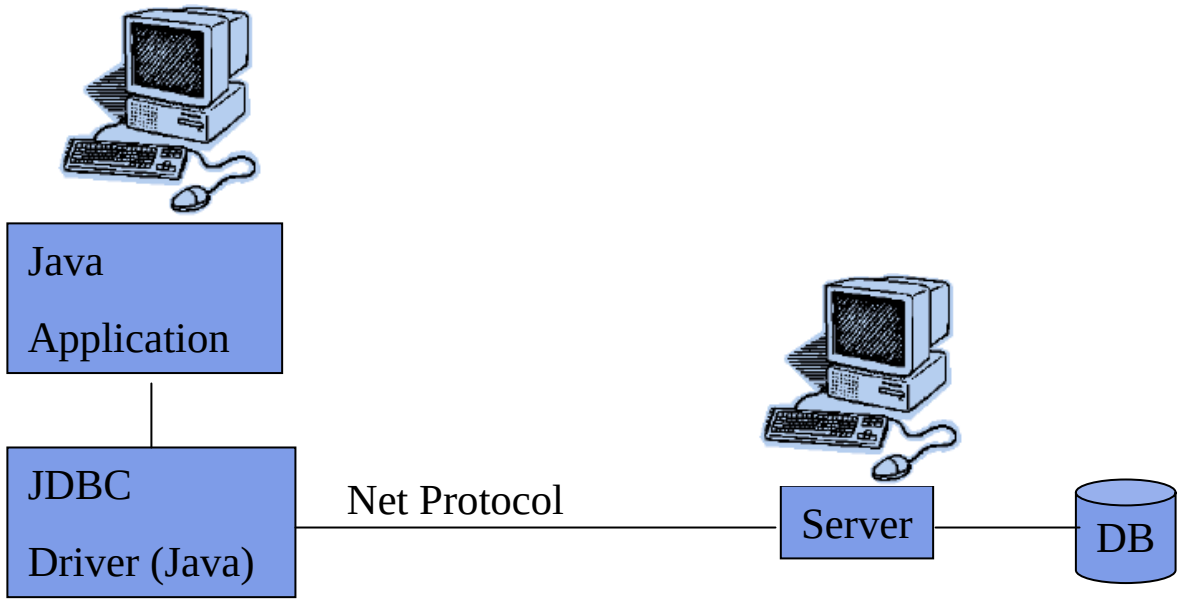
JDBC Driver Types

- **Type 2: Native API / Partially Java**



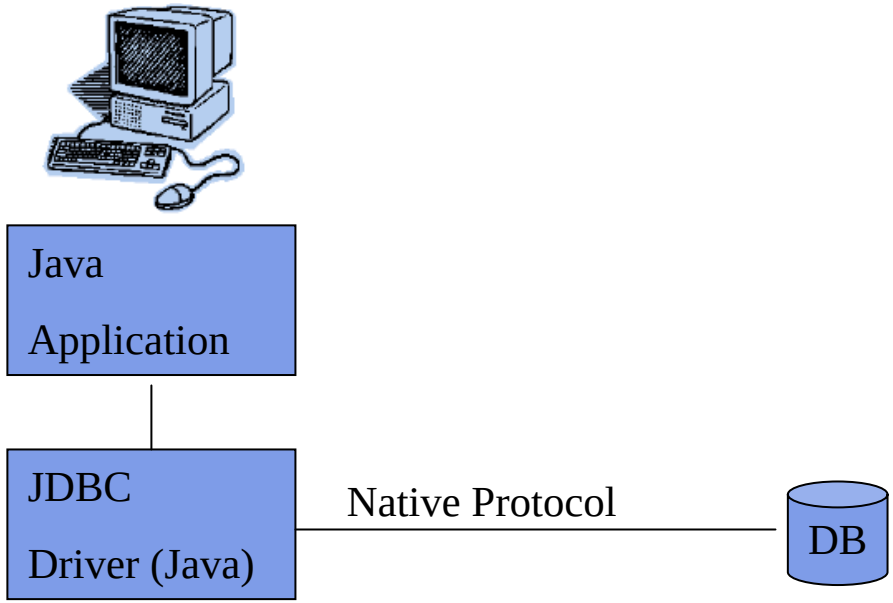
JDBC Driver Types

- **Type 3: Pure Java / Net Protocol.**

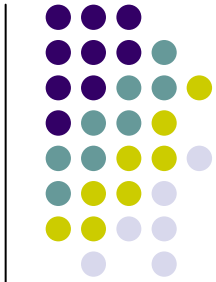


JDBC Driver Types

- **Type 4: Pure Java / Native Protocol.**

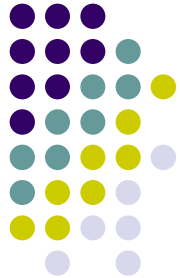


Eclipse JDBC setup



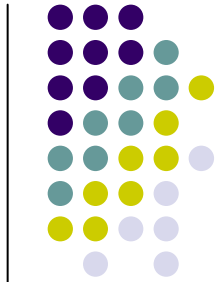
- Install driver
  - Download MySQL JDBC driver from the Web
    - http://dev.mysql.com/downloads/connector/j/5.0.html
  - Unzip mysql-connector-xxx.jar
  - Add mysql-connector-xxx.jar to Eclipse project
    - Project → Properties → Java Build Path → Libraries → Add External JARs

JDBC steps



1. Connect to database
2. Query database (or insert/update/delete)
3. Process results
4. Close connection to database

1. Connect to database



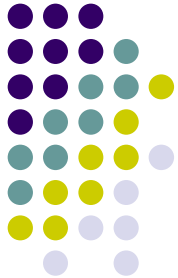
- Load JDBC driver
  - `Class.forName("com.mysql.jdbc.Driver").newInstance();`
- Make connection
  - `Connection conn = DriverManager.getConnection(url);`
- URL
  - **Format: "jdbc:<*subprotocol*>:<*subname*>"**
  - `jdbc:mysql://localhost/systemsDB`

2. Query database

a. Create statement
   - `Statement stmt = conn.createStatement();`
   - stmt object sends SQL commands to database
   - Methods
     - executeQuery() for SELECT statements
     - executeUpdate() for INSERT, UPDATE, DELETE, statements

b. Send SQL statements
   - stmt.executeQuery(“SELECT …”);
   - stmt.executeUpdate(“INSERT …”);

3. Process results



- Result of a SELECT statement (rows/columns) returned as a ResultSet object
  - `ResultSet rs = stmt.executeQuery("SELECT * FROM users");`
- Step through each row in the result
  - `rs.next()`
- Get column values in a row
  - `String userid = rs.getString(“userid”);`
  - `int type = rs.getInt(“type”);`

users table				
userid	**firstname**	**lastname**	**password**	**type**
Bob	Bob	King	cat	0
John	John	Smith	pass	1

Queries

- **Result Set Cursor:**

EMPLOYEE	
NAME	SALARY
Yossi	15000
Miri	15000

Initial Cursor position

Cursor <u>after</u> first call to *rs.next()*

Print the users table

```
ResultSet rs = stmt.executeQuery("SELECT * FROM users");

while (rs.next()) {
    String userid = rs.getString(1);
    String firstname = rs.getString(“firstname”);
    String lastname = rs.getString(“lastname”);
    String password = rs.getString(4);
    int type = rs.getInt(“type”);
    System.out.println(userid + ” ” + firstname + ” ” +
        lastname + ” ” + password + ” ” + type);
}
```

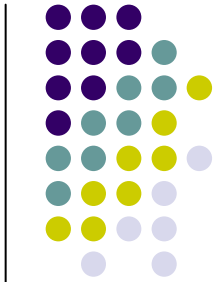
users table				
userid	**firstname**	**lastname**	**password**	**type**
Bob	Bob	King	cat	0
John	John	Smith	pass	1

Add a row to the users table

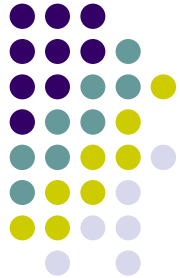
```
String str =
     "INSERT INTO users
      VALUES('Bob', 'Bob', 'King',
             'cat', 0)”;

// Returns number of rows in table
int rows = stmt.executeUpdate(str);
```

users table				
userid	**firstname**	**lastname**	**password**	**type**
Bob	Bob	King	cat	0

4. Close connection to database

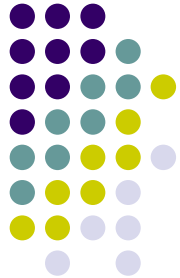


- Close the ResultSet object
  - `rs.close();`
- Close the Statement object
  - `stmt.close();`
- Close the connection
  - `conn.close();`



```
import java.sql.*;

public class Tester {
   public static void main(String[] args) {
      try {
          // Load JDBC driver
          Class.forName("com.mysql.jdbc.Driver").newInstance();

          // Make connection
          String url =
                "jdbc:mysql://128.100.53.33/GRP?user=USER&password=PASS"
          Connection conn = DriverManager.getConnection(url);

          // Create statement
          Statement stmt = conn.createStatement();

          // Print the users table
          ResultSet rs = stmt.executeQuery("SELECT * FROM users");
          while (rs.next()) {
                ...
          }

          // Cleanup
          rs.close(); stmt.close(); conn.close();

   } catch (Exception e) {
          System.out.println("exception " + e);
   }
}
```

Queries: Joins



- **Joining tables** with similar column names:
- You *may* need to read columns by index.

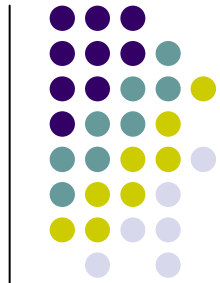
product	
NAME	**ID**
Tomatoes	1
Shampoo	2

supplier	
NAME	**PROD_ID**
Farmer1	1
Hawaii	2

JOIN	
product.NAME	**supplier.NAME**
Tomatoes	Farmer1
Shampoo	Hawaii

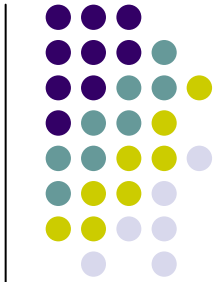
```
ResultSet rs = stmt.executeQuery(
    “select p.NAME,s.NAME from PRODUCT p, SUPPLIER s  where s.PROD_ID=p.ID “);
while(rs.next())
   System.out.println( rs.getString(1) +  “  “ + rs.getString(2));
```

Transactions



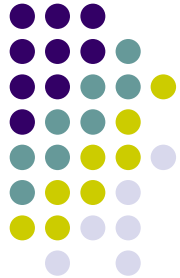
- Currently every executeUpdate() is “finalized” right away
- Sometimes want to a set of updates to all fail or all succeed
  - E.g. add to Appointments and Bookings tables
  - Treat both inserts as one transaction
- Transaction
  - Used to group several SQL statements together
  - Either all succeed or all fail

Transactions



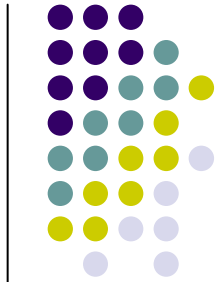
- Commit
  - Execute all statements as one unit
  - “Finalize” updates
- Rollback
  - Abort transaction
  - All uncommited statements are discarded
  - Revert database to original state

Transactions in JDBC



- Disable auto-commit for the connection
  - conn.setAutoCommit(false);
- Call necessary executeUpdate() statements
- Commit or rollback
  - conn.commit();
  - conn.rollback();

Prepared Statements



- **A prepared statement** is pre-compiled only once.
- **Allows arguments** to be filled in.
- **Useful for:**
  - Efficiency.
  - Convenience.
  - Handling special types (e.g. long binary data).
  - Security (reduces danger of SQL injection).

Prepared Statements

- **Example:**

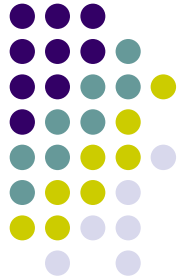
```
Class.forName(myDriverName);
Connection con=DriverManager.getConnection(myDbUrl, “john”, “secret”);
PreparedStatement stmt=con.prepareStatement(“insert into EMPLOYEE values(?,?)”);

stmt.setString(1, “john”);
stmt.setDouble(2, 12000);
stmt.executeUpdate();

stmt.setString(1, “paul”);
stmt.setDouble(2, 15000);
stmt.executeUpdate();
…
… // close resources
```

param # 1

param # 2

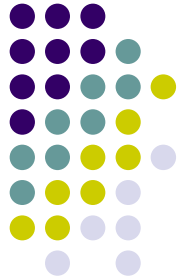
Fill in params

Callable Statement



- **Let us demonstrate:**
- Defining a stored procedure through java (vender-specific).
- Invoking a stored procedure.
- Note: not all drivers support these features.

Callable Statement



- **Defining a stored procedure/function.**
- Vendor-specific code.

```
Connection con= …
Statemenet stmt=con.createStatement();
String str="create function countNames (empName VARCHAR)  RETURN NUMBER  " +
          " IS  cnt  NUMBER  " +
          "BEGIN  " +
          "select count(*) INTO cnt from EMPLOYEE where name=empName  " +
          "return cnt; " +
          "END countNames"  ;
stmt.executeUpdate(str);
```

Callable Statement

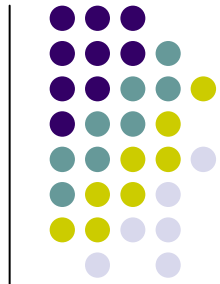
- **Invoking a Stored Function, Using CallableStatement:**

```
CallableStatement cs= con.prepareCall( "{?=call SALES.countNames (?)}" );
cs.registerOutParameter(1, Types.INTEGER);
cs.setString(2, 'john');
cs.execute();
System.out.println( cs.getInt(1));
```

Param #1 :
Out, integer

Param #2:
In, String

Advanced Features



- Many features were added to the JDBC standard (currently JDBC 4.0)
  - http://java.sun.com/products/jdbc/download.html
- There are other (competing or complementary) technologies:
  - JDO
  - Entity EJB's
  - Hibernate
  - More…

References

- Some slide content borrowed from
  - http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html
  - http://otn.oracle.co.kr/admin/seminar/data/otn-jdbc.ppt
  - http://notes.corewebprogramming.com/student/JDBC.pdf