

Java and Database Connectivity

Thava Alagu

thavamuni.alagu@sun.com

Database Overview

Common Database Models

- **Hierarchical Model**

- > Tree like structures like filesystems; eg: IBM's IMS

- **Network Model**

- > Can access many-many relations with pointers like.
 - e.g. CA-IDMS (CODASYL network std)

- **Relational Model**

- > Set of Tables. Supports declarative query language like SQL.

- **Object Relational Model**

- > Relational + user defined data type & methods

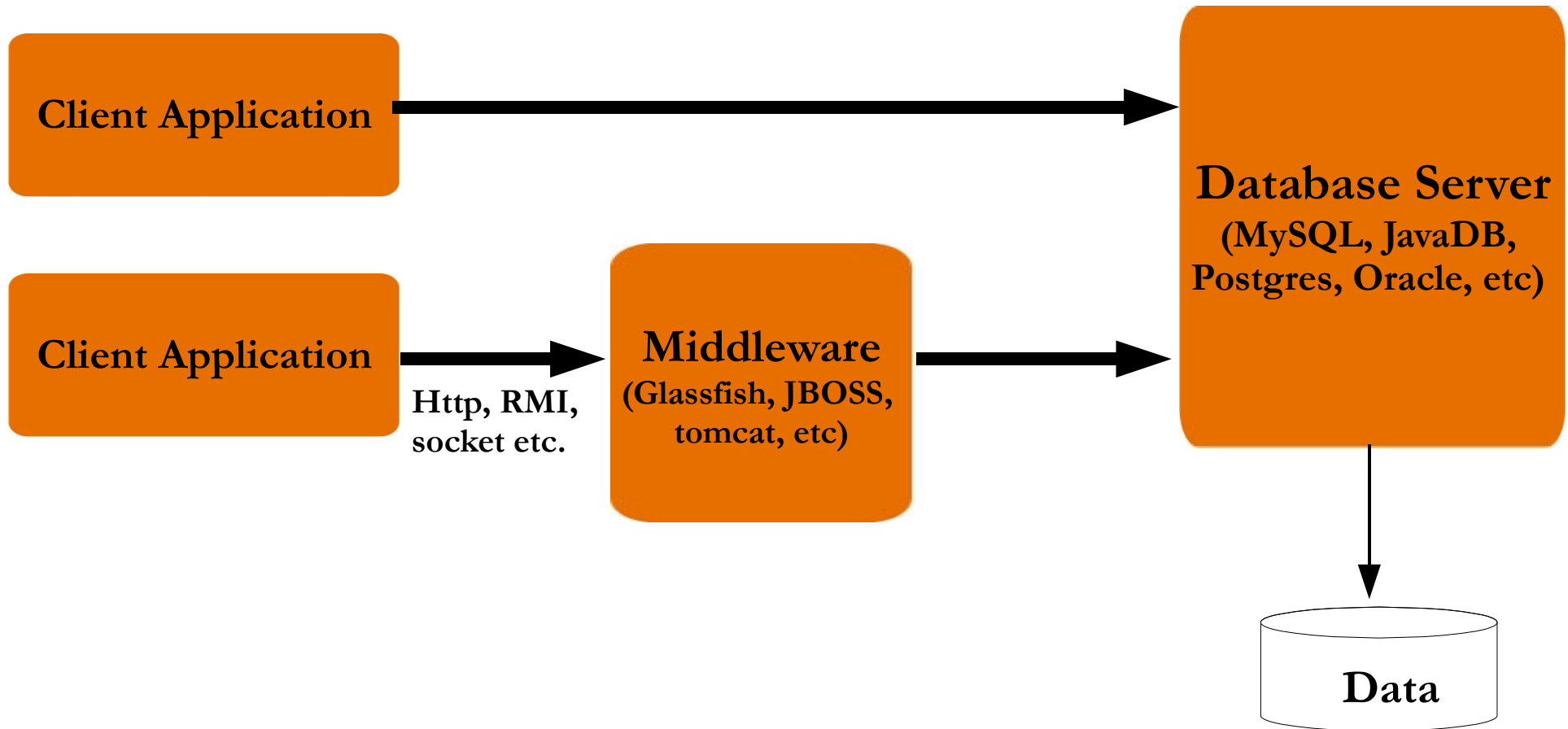
ACID Properties Of Transactions

- **Atomic**
- **Consistent**
- **Isolated**
- **Durable**

Relational Model ...

- **Set Oriented**
- **Based on Relational Algebra**
- **Relations and Tuples**
 - Tables and Columns
- **SQL is the most famous QL.**
 - Declarative – not procedural
 - Select, insert, union, intersect etc.
- **Associations with primary/foreign keys**
- **Lookup optimizations using index**
- **Semantic consistency using constraints**

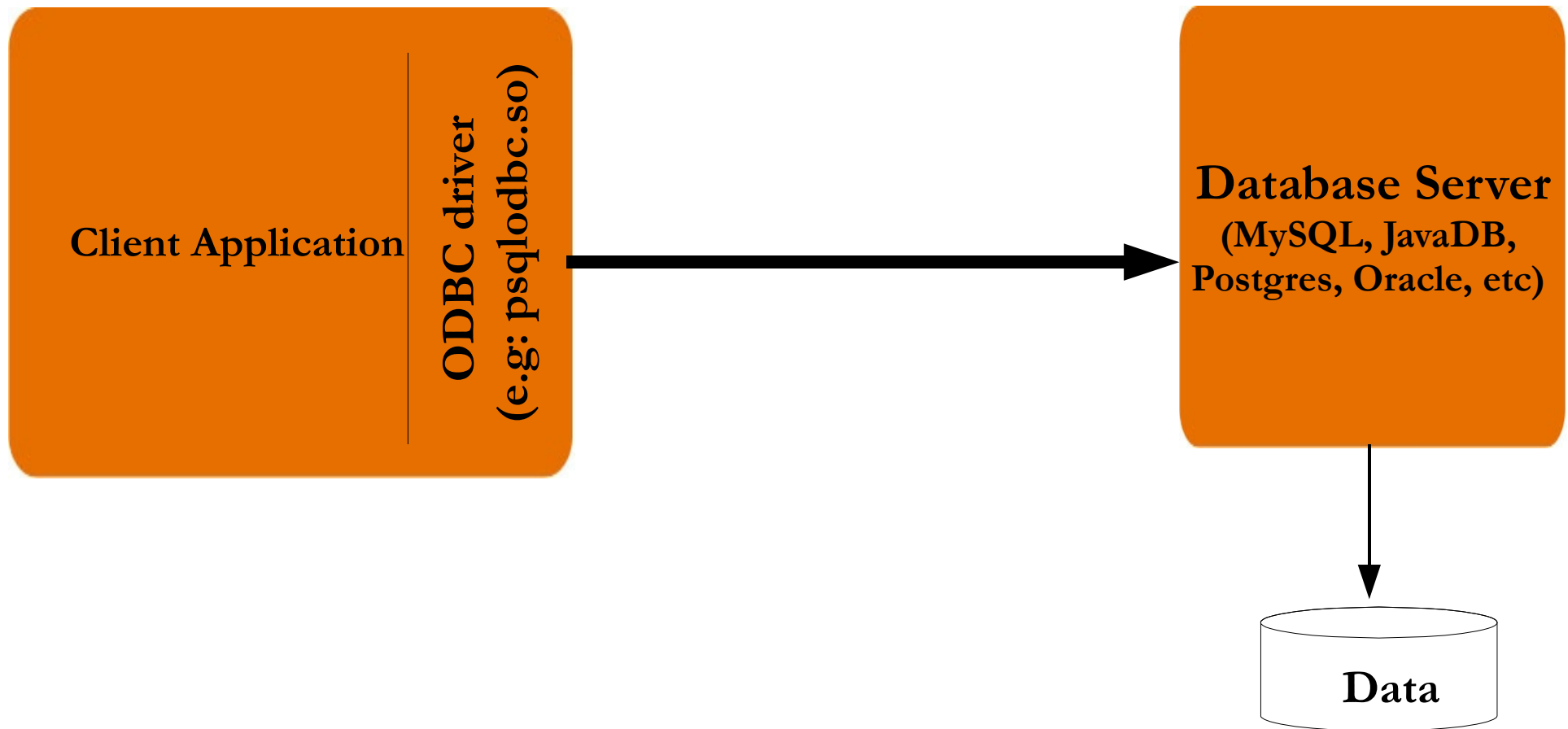
Connecting to Database ...



What is ODBC ?

- Open Database Connectivity
- Standard programming API for data access
- Database vendor provides ODBC Driver
 - > Example: `psqlodbc.so` is odbc driver for postgres
- Client program does not need recompile
- Client is database independent
- Client is still OS dependent

ODBC Connectivity



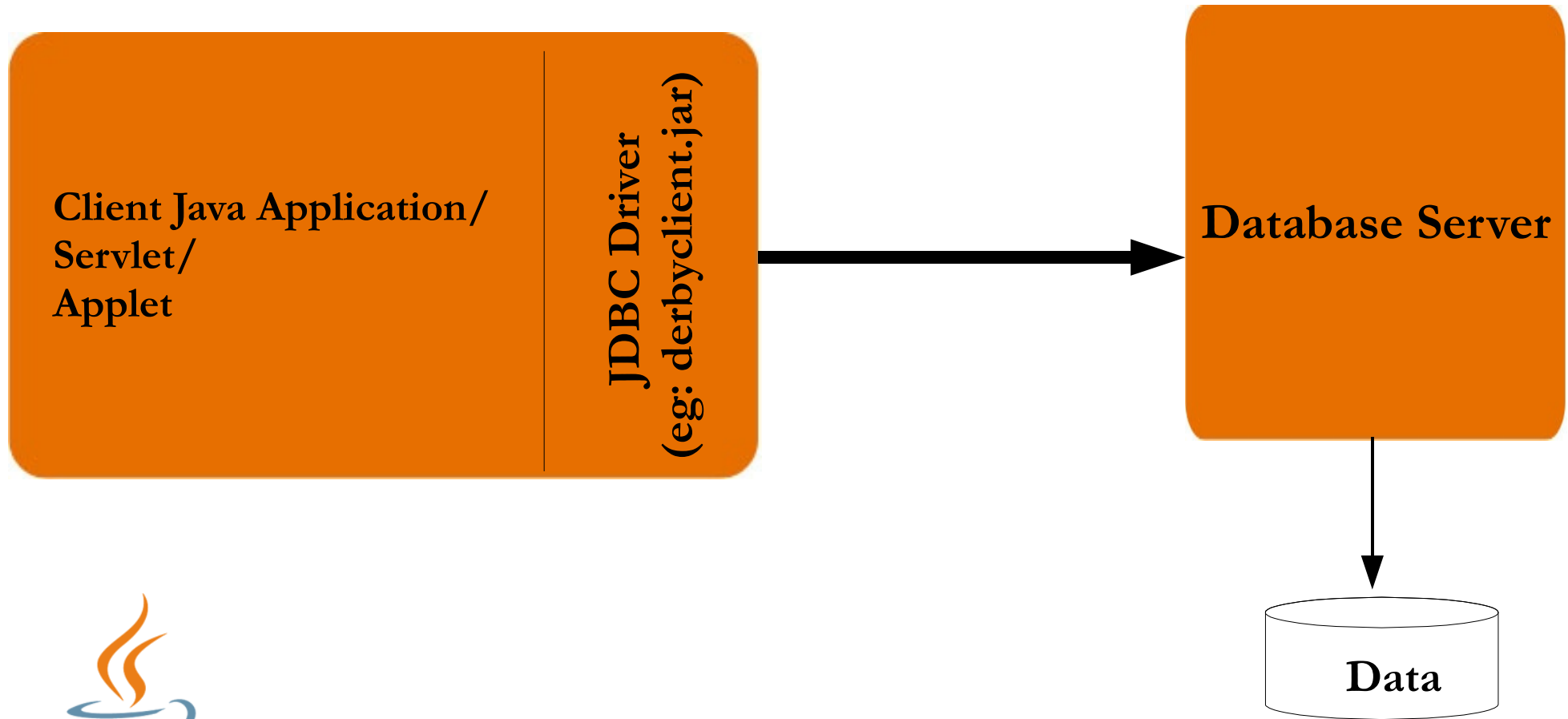
Client is database independent but OS dependent.

What is JDBC ?

- Java Database Connectivity
- Java programming API for data access
- Database vendor provides JDBC Driver
 - > Example: derbyclient.jar for JavaDB
- Client is Database & OS independent!



JDBC Connectivity



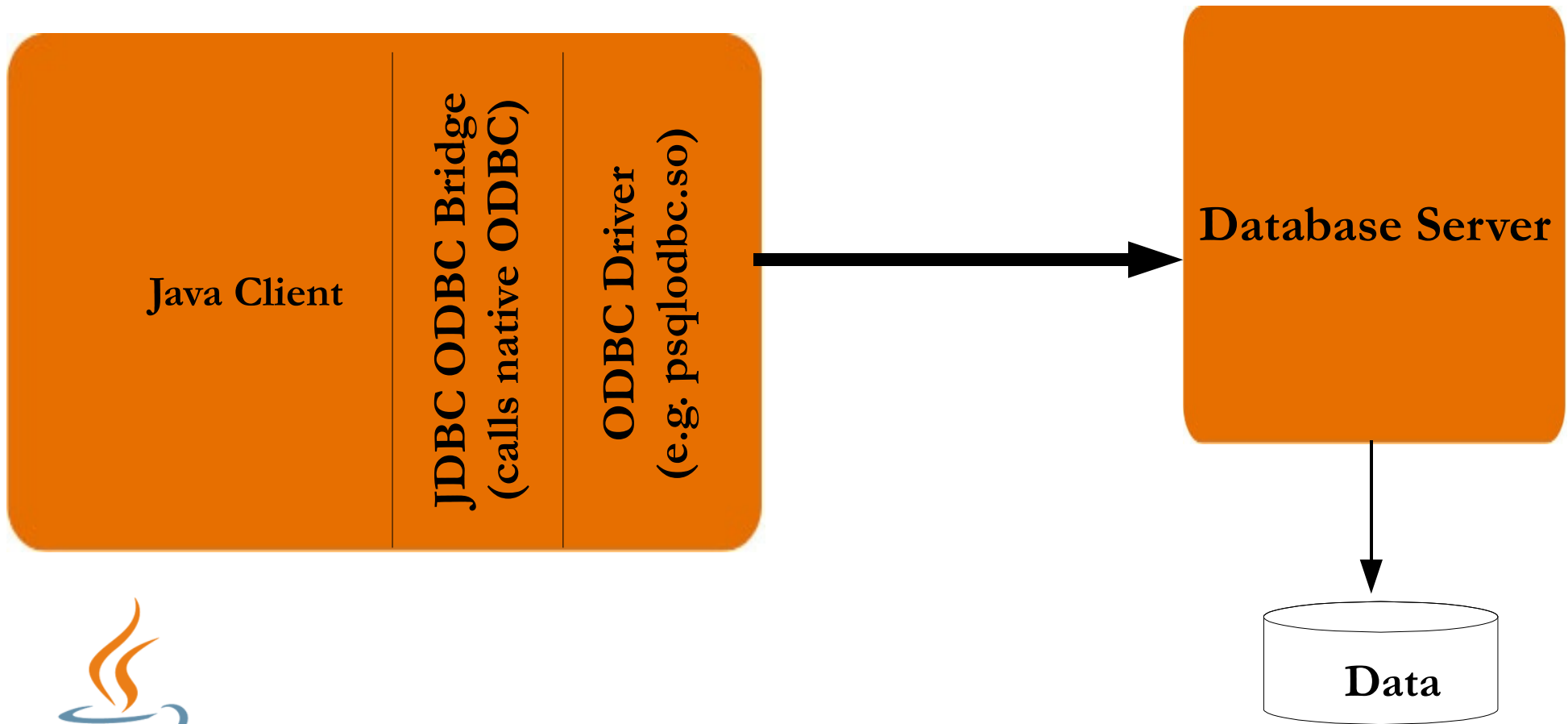
Client is database and OS independent !

Types Of JDBC Drivers ...

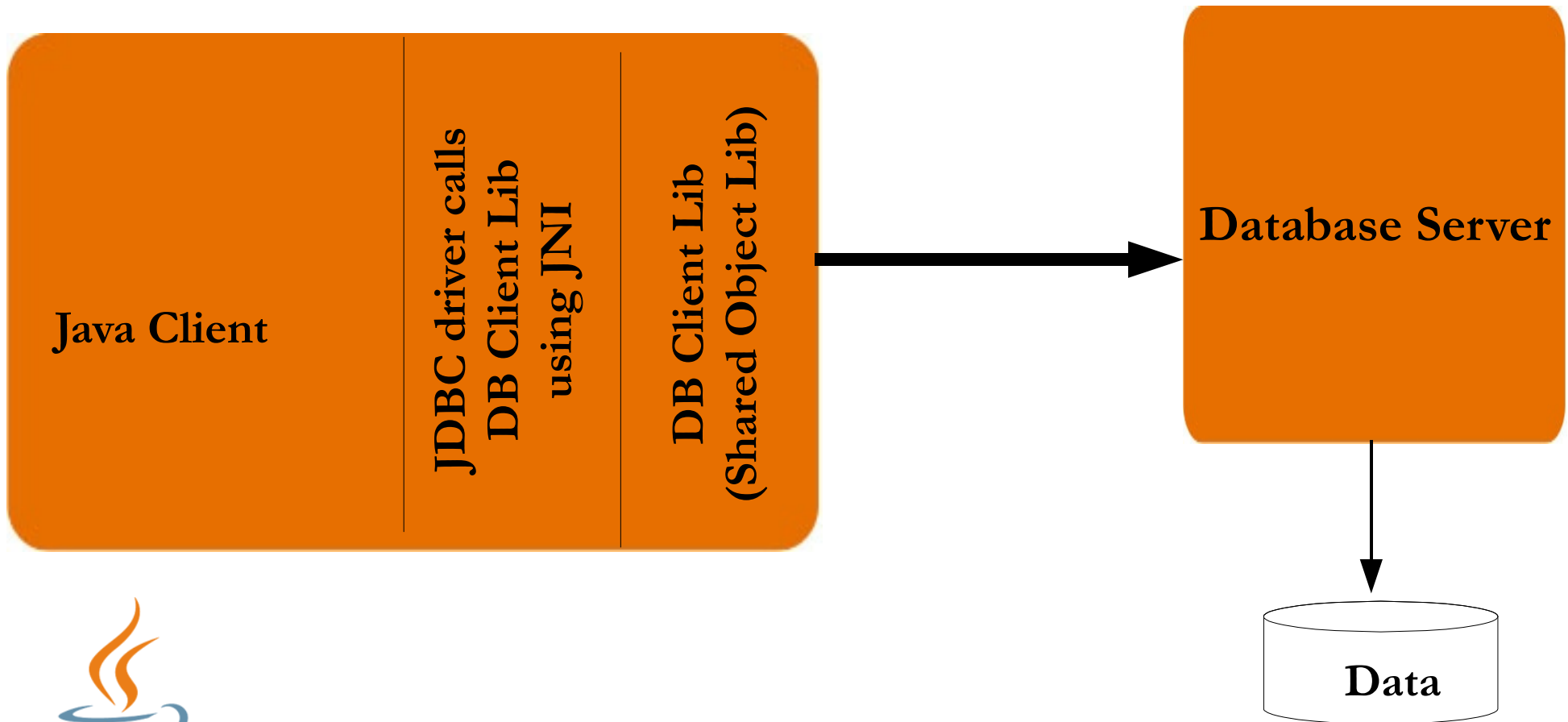
- Type 1: JDBC-ODBC Bridge Driver
- Type 2: Part Java + Part JNI
- Type 3: Pure Java, talking to Middleware
- Type 4: Pure 100% Java



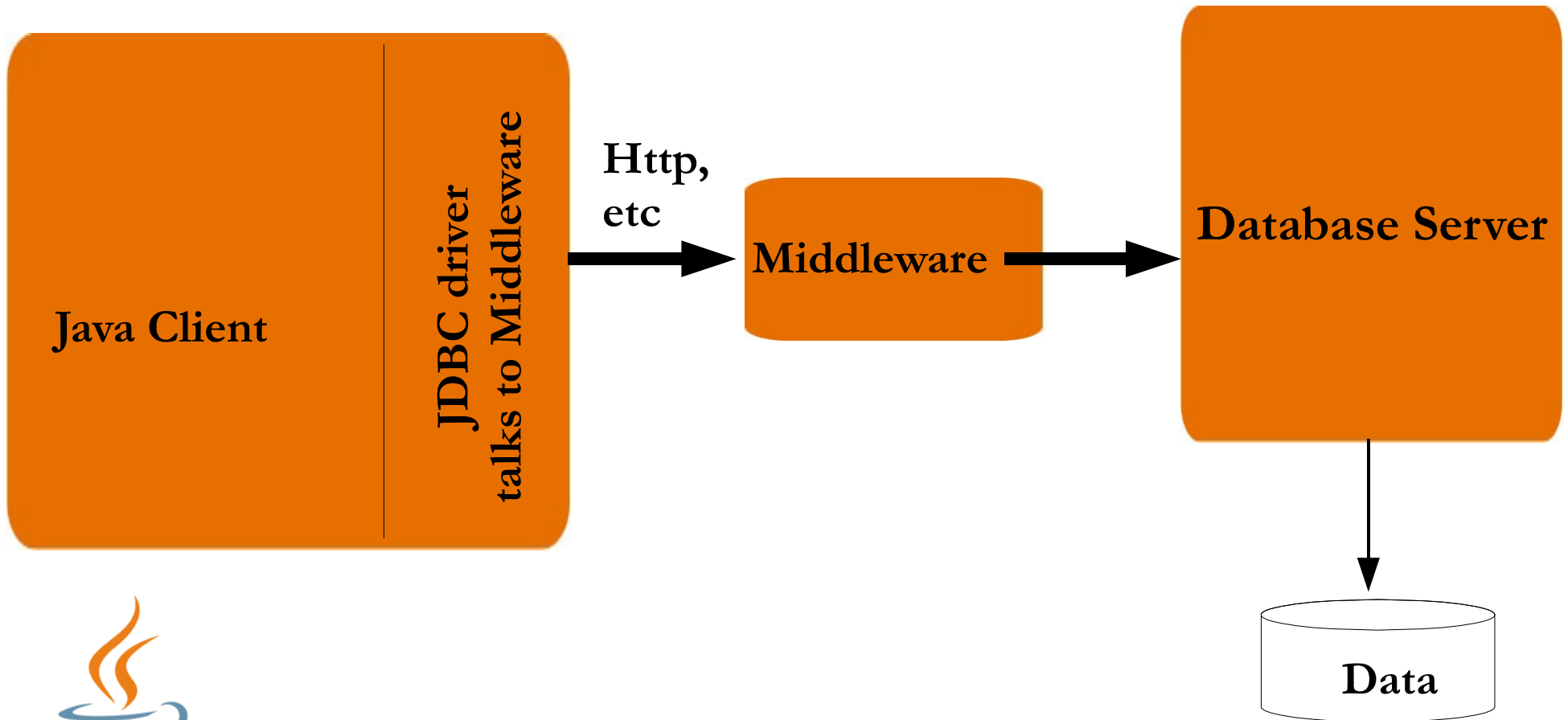
Type 1 JDBC Driver



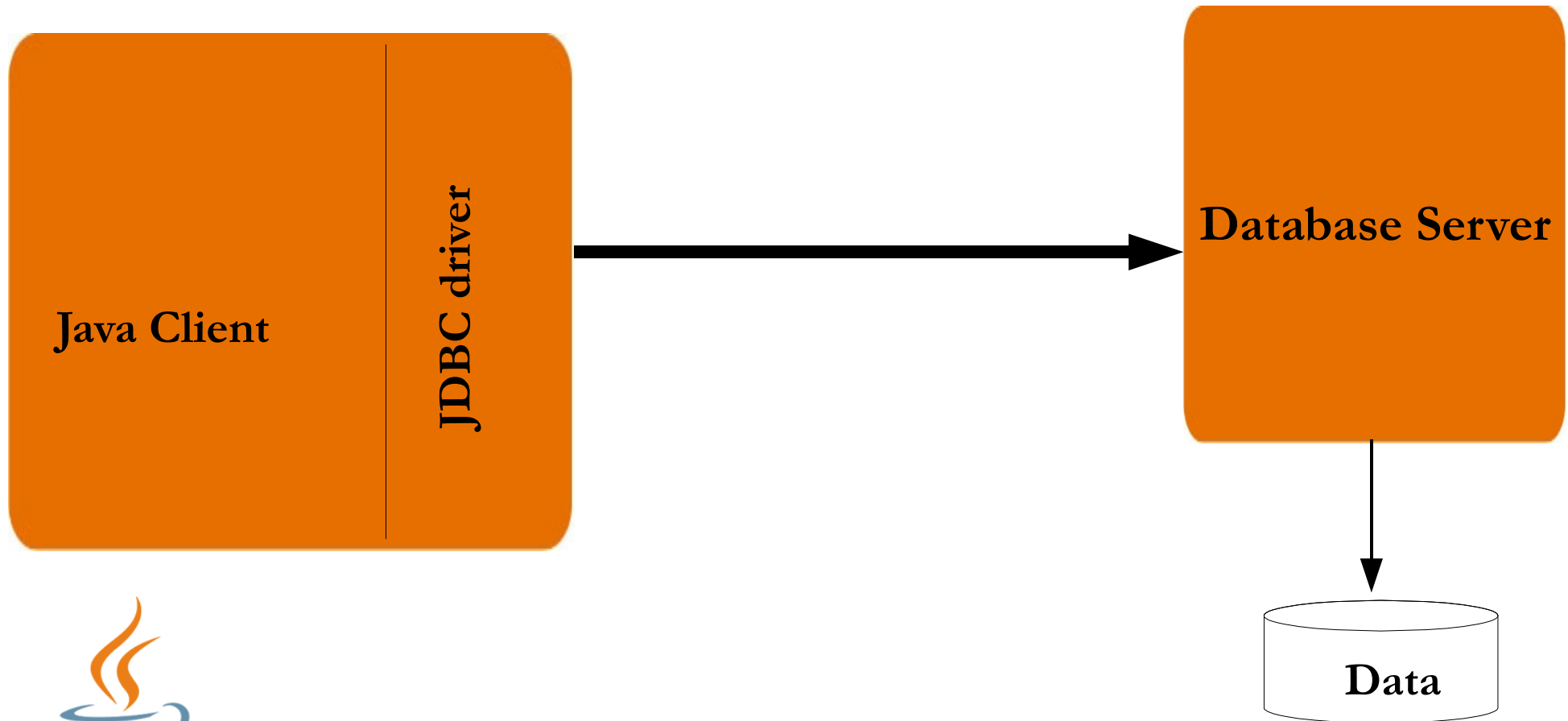
Type 2 JDBC Driver



Type 3 JDBC Driver



Type 4 JDBC Driver



“100% Java!”

JDBC Programming ...

- Load JDBC Driver
- Make Connection
- Execute SQL and get result
- Process ResultSet
- Have a cup of hot coffee!



JDBC Basic Java Classes ...

- DriverManager
- Connection
- Statement
- ResultSet
- DatabaseMetaData
- ResultSetMetaData
- SQLException



Simple Example ...

```
import java.sql.*;
```

```
String driver = "org.apache.derby.jdbc.EmbeddedDriver";
```

```
String protocol = "jdbc:derby:myDB;create=true";
```

```
Class.forName(driver);
```

```
Connection conn = DriverManager.getConnection(protocol);
```

```
Statement s = conn.createStatement();
```

```
s.execute("create table movie (name varchar(30), actor varchar(30))");
```

```
s.execute("insert into movie values ('vanilla sky', 'tom cruise')");
```

```
ResultSet rs = s.executeQuery("select * from movie");
```

```
while (rs.next()) {
```

```
    System.out.println("Movie Name: " + rs.getString(1) +  
        " ; Actor: " + rs.getString(2));
```

```
}
```

Exercise : JavaDB Demo



On your opensolaris developer express system follow these steps :

```
$ bash
$ export DERBY_HOME=/opt/SUNWjavadb
$ cd /opt/SUNWjavadb/bin
$ . setEmbeddedCP ; # This sets proper class path
# Above step is same as:
$ export CLASSPATH=$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools.jar:
# Now cd to your directory where you have write permission
$ /opt/SUNWjavadb/bin/ij      # This is command line SQL interface
    create database mydb; create table mytable(i integer); insert into mytable values(1); quit;

# This example runs simple JDBC program using Embedded JavaDB driver.
$ cd /opt/SUNWhadb/demo/programs/simple
$ javac SimpleApp.java
$ java  SimpleApp

# Now run JavaDB in Client/Server mode ...
$ . /opt/SUNWhadb/bin/setNetworkServerCP ; # This sets proper CLASSPATH
$ . /opt/SUNWhadb/bin/startNetworkServer

# Now compile and run programs in /opt/SUNWjavadb/demo/programs/nserverdemo
```



Note: JavaDB is the name of the distribution of Derby Database.

JDBC Programming

Digging Deeper



Driver Manager

```
String driver = "org.apache.derby.jdbc.ClientDriver";  
String protocol = "jdbc:derby://localhost:1527/Mydb;create=true;";  
Class.forName(driver);  
Connection conn = DriverManager.getConnection(protocol);
```

- Class.forName() dynamically loads driver on need basis.
- As part of loading, each driver creates an instance of itself and registers with DriverManager by calling DriverManager.RegisterDriver();
- Calls drivers getConnection() method on demand
- Implements late binding of driver for client program
- Driver name typically configured in properties file
- DriverManager is singleton class
 - > All methods are static
- Can manage multiple JDBC Drivers in client program.

Specifying JDBC Drivers ...

```
String driver = "org.apache.derby.jdbc.ClientDriver";  
Class.forName(driver);
```

- Can be explicitly loaded by **Class.forName()**;
- Define **jdbc.drivers** system property.
 - > e.g. Java **-Djdbc.drivers=org.mydriver:org.yourdriver MyApp**
 - > All specified drivers will be loaded on the first call to DriverManager
- Set the system property **jdbc.drivers**
 - > `System.setProperty("jdbc.drivers", "org.apache.derby.jdbc.EmbeddedDriver");`
 - > DriverManager loads drivers only once, so this must be done before calling any DriverManager methods.

JDBC URL ...

```
String protocol = "jdbc:derby://localhost:1527/Mydb;create=true;";  
Connection conn = DriverManager.getConnection(protocol);
```

- Format : `<protocol>:<subprotocol>:<resource>`
 - > Example: main protocol is jdbc; subprotocol is derby;
 - > Rest is interpreted specific to subprotocol (i.e. Derby)
 - > Resource format is implementation dependent, but in general follows this syntax:
 - > `//hostname:portname/Dbname`

Connection

```
String protocol = "jdbc:derby://localhost:1527/Mydb;create=true;";  
Connection conn = DriverManager.getConnection(protocol);
```

- JDBC is Session Oriented Protocol. Remember to Close!
- Use one of 3 methods:

```
public static Connection getConnection(String url)  
public static Connection getConnection(String url, Properties info)  
public static Connection getConnection(String url, String user, String password)
```
- Examples:
 - > `DriverManager.getConnection(protocol, "myuser", "mypassword");`
 - > `String protocol = "jdbc:derby:Mydb;create=true;username=myuser;password=mypass";`
 - > Properties props;
 - > `props.setProperty("user", "myuser");`
 - > `props.setProperty("password", "mypwd");`
 - > `DriverManager.getConnection(url, props);`

Statement

```
Statement s = conn.createStatement();  
s.execute(...);
```

- Abstraction for SQL statement
- Three types of Statement :
 - > Statement
 - > PreparedStatement
 - > CallableStatement
- Remember to close statement and release resources.

Executing Queries ...

```
Statement s = conn.createStatement();
```

```
s.execute("select * from mytable; select count(*) from mytable");  
while ( stmt.getMoreResults() || (stmt.getUpdateCount() != -1))  
    { ResultSet rs = s.getResultSet(); ... ; }
```

```
ResultSet rs = s.executeQuery("select * from mytable");
```

```
s.executeUpdate("update mytable set quantity = 20");
```

- `boolean execute(String)` can return multiple result sets!
- `executeQuery()` returns single `ResultSet`.
- `executeUpdate()` is for DDL or insert/update statement.

Prepared Statement

- **Example**

```
// Creating a prepared Statement  
String sqlString = "UPDATE Accounts SET bonus = ? where name = ?";  
PreparedStatement ps = connection.prepareStatement(sqlString);  
ps.setInt(1, 100); // Sets first argument  
ps.setString(2, "Thava"); // Sets second argument  
ps.executeUpdate(); // Executes the update
```

- **You can reuse precompiled statement for executing multiple times.**

Callable Statements

```
Statement stmt = con.createStatement();  
stmt.executeUpdate("create procedure SHOW_SUPPLIERS ... ");  
CallableStatement cs = con.prepareCall("{call SHOW_SUPPLIERS}");  
ResultSet rs = cs.executeQuery();
```

- Callable Statements Used to call Stored Procedures in the database
- Stored procedures are written using SQL and/or other languages.
- Example: PL/SQL for Solaris, PL/pgSQL for Postgres
- Provides reuse of optimized code running at server side.

Support For Savepoints

```
Statement stmt = conn.createStatement();
int rows = stmt.executeUpdate("...");
// set savepoint
Savepoint svpt1 = conn.setSavepoint("SAVEPOINT_1");
rows = stmt.executeUpdate("...");
...
conn.rollback(svpt1);
...
conn.commit();
```

- Can take snapshots within transaction and do partial rollback.
- Can be used to simulate nested transaction like semantics.

Scrollable/Updatable ResultSet

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
                                     ResultSet.CONCUR_UPDATABLE);  
ResultSet uprs = stmt.executeQuery("SELECT * FROM movies");
```

- The default ResultSet you get from no-argument createStatement() is :
 - > Non-scrollable (Moves only forward)
 - > Not Updateable (Read only, No write)
 - Statement createStatement(int resultSetType, int resultSetConcurrency)
 - resultSetType specifies whether it is scrollable:
 - ResultSet.TYPE_FORWARD_ONLY
 - ResultSet.TYPE_SCROLL_INSENSITIVE (Unaffected by changes to underlying database)
 - ResultSet.TYPE_SCROLL_SENSITIVE (Reflects changes to underlying database)
 - ResultSetConcurrency determines whether data is updatable. Its possible values are
 - CONCUR_READ_ONLY
 - CONCUR_UPDATABLE
- Note : Not all databases support these options.

Scrollable ResultSet

- You can use following functions on a Scrollable ResultSet :
 - boolean next(), boolean previous(), boolean first(), boolean last()
 - void afterLast(), void beforeFirst()
 - boolean isFirst(), boolean isLast(), boolean isBeforeFirst(), boolean isAfterLast()

Closing connections and Statements ...

```
try {  
    Connection con =  
        DriverManager.getConnection(url);  
    Statement s = con.createStatement();  
    // execute query etc.  
} catch (SQLException sqle) {  
    sqle.printStackTrace();  
} finally {  
    try { s.close(); con.close(); }  
    catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Use finally clause to ensure releasing of resources.

Other JDBC Features ...

- RowSet provides Serializable alternative to ResultSet
- DataSource, ConnectionPoolDataSource, PooledConnection
- XADataSource, XAConnection, XAResource
- Interface to set Isolation Levels
 - > Read Uncommitted, Read Committed, Repeatable Read, Serializable.
- Support Blob, Clob data types :
 - > Binary/Character Large Objects
- Supports XML Datatype

Handling SQLExceptions ...

```
try {  
    // Code that could generate an exception goes here.  
    // If an exception is generated, the catch block below  
    // will print out information about it.  
}  
catch(SQLException ex) {  
    System.out.println("\n--- SQLException caught ---\n");  
    while (ex != null) { // Note: while loop!!!  
        System.out.println("Message: "  
            + ex.getMessage ());  
        System.out.println("SQLState: "  
            + ex.getSQLState ());  
        System.out.println("ErrorCode: " // Vendor specific  
            + ex.getErrorCode ());  
        ex = ex.getNextException();  
        System.out.println("");  
    }  
}
```

Handling Warnings ...

```
SQLWarning warn = rs.getWarnings();
if (warn != null) {
    System.out.println("\n---Warning---\n");
    while (warn != null) {
        System.out.println("Message: "
            + warn.getMessage());
        System.out.println("SQLState: "
            + warn.getSQLState());
        System.out.print("Vendor error code: ");
        System.out.println(warn.getErrorCode());
        System.out.println("");
        warn = warn.getNextWarning();
    }
}
```

Related Technologies ...

SQLJ – Embedding SQL in Java

```
#sql [ctx] {  
    INSERT INTO DSN8710.EMP  
        (EMPNO, FIRSTNME, MIDINIT, LASTNAME, HIREDATE, SALARY)  
        VALUES (:empno, :firstname, :midinit, :lastname, CURRENT DATE, :salary)  
};  
#sql private static iterator EmployeeIterator(String, String, BigDecimal);  
...  
EmployeeIterator iter;  
#sql [ctx] iter = {  
    SELECT LASTNAME , FIRSTNME , SALARY FROM DSN8710.EMP  
        WHERE SALARY BETWEEN :min AND :max  
};  
while (true) {  
    #sql {  
        FETCH :iter INTO :lastname, :firstname, :salary  
    };  
    if (iter.endFetch()) break;  
    // Print row...  
}  
iter.close();
```

SQLJ

- Preprocesses Java code to translate embedded SQL to Java
- Typical implementations use translated Java Uses JDBC library internally

SQL Standards

- 1986 : SQL-86 First published by ANSI.
 - Ratified by ISO in 1987. Also known as SQL-87
- 1989 : SQL-89 Minor Revision
- 1992 : SQL-92 Major Revision. Also known as SQL-2
 - Date, Time, Check constraints, Alter, Scrolling cursors, etc
- 1999 : SQL:1999 [aka SQL-3]
 - Added regular expression matching, recursive queries, triggers, support for procedural statements, non-scalar types, and some object-oriented features. ...
- 2003 : SQL:2003
 - Introduced XML-related features, window functions, standardized sequences, and columns with auto-generated values (including identity-columns). ...
- 2006: SQL:2006
 - Importing/Exporting XML data; publishing SQL data in XML format; XQuery support functions

JDBC Specification History

JDBC 1.0

- JDBC 1.0.2 API : 1997 (Jan) :
 - > Based on X/Open SQL CLI (Call Level Interface) – [ODBC too]
 - > Supports Min ANSI SQL92 (aka SQL2)
 - > Fairly sophisticated and includes most important classes.
 - > API Specification: <http://java.sun.com/products/jdbc/jdbc-spec-0120.pdf>

JDBC 2.1

- JDBC 2.1 API : 1999 (Oct)
 - > Supports subset of SQL1999
 - > Userdefined Datatype, Scrollable Cursor, BLOBS, Datasource,
 - > REF Type, Distinct Type (eg: Numeric(10,2))
 - > Uses more of JavaBeans; More in line with JTS, JNDI, EJB
 - > Batchupdates (process multiple Resultsets)
 - > Serializable Rowset Bean, JNDI lookup of Driver Vs DriverManager
 - > Hook to ConnectionPooling
 - > Distributed Transaction support (as supported by JTS API)

JDBC 3.0

- JDBC 3.0 : 2001 Oct
 - > More of SQL:1999 features, align with J2EE
 - > Savepoints
 - > More connectionpool config
 - > Blob & Clob partial update
 - > Retrieve REF type
 - > More minor features

JDBC 4.0

- JDBC 4.0 : 2006 Nov
 - > Target SQL:2003
 - > SQLXML Object
 - > Support Escape syntax

Pointers

- JDBC Tutorial: <http://java.sun.com/docs/books/tutorial/jdbc/index.html>
- JDBC Spec : <http://java.sun.com/products/jdbc/download.html>
- Sun India University Programs:
 - > <http://sunweb.india.sun.com/univ>
- Java Technology Colloboration : Forums, etc :
 - > <http://java.net/>
- Effective Java Programming: Book by Joshua Bloch
- Student Developers:<http://developers.sun.com/students/>
- OpenSolaris: <http://opensolaris.org>
- Netbeans: <http://www.netbeans.org>
- Sun Developer Network: <http://developers.sun.com>



More Pointers

- MySQL : <http://mysql.com/>
- JavaDB : <http://developers.sun.com/javadb/>
 - <http://db.apache.org/derby/index.html>
- PostgreSQL : <http://www.postgresql.org>
- Free developer tools: <http://www.sun.com/edu/edusoft>
- Free training on Sun technologies: <http://sunacademic.com>
- Sun Developer Network : <http://developers.sun.com>
- Join the Student Connection
 - > <http://www.sun.com/emrkt/edu/studentconnection/>



Q & A

