Session 1:

JDBC
Java Database Connectivity

Contents

- Database Access Methods, ODBC, JDBC,JDBC Architecture
- The java.sql package
- DriverManager, Driver, Connection, Statement, ResultSet
- Writing database applications

Why Database Access Methods need?

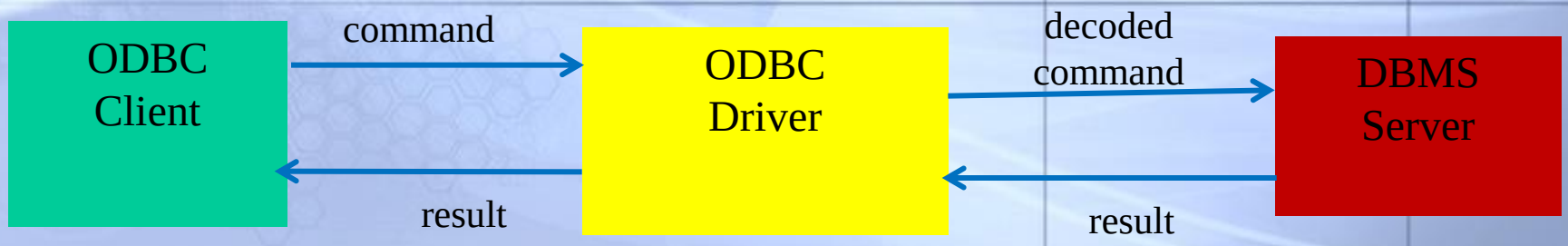
- Databases help in easy retrieval and processing of data.
- In order to update or query the database from applications, you will need to learn about SQL processing in DBMS.
- This can be a tedious task for the application developer.
- The solution is to build applications that will act as an interface between the application and the database.
- The applications will accept values from the end user by using user-friendly GUIs, and update or query the database.

What is ODBC?

- abbreviation for **Open Database Connectivity.**
- an interface to access databases via SQL queries.
- can be used as an access tool to various databases :
  - MS-Access, dBase, DB2, Excel, etc..
- was first created by Microsoft and Simba Technologies.

How ODBC is Processed

- To use the ODBC, three components are needed:
  1. ODBC client,
  2. ODBC driver, and
  3. DBMS server (ex. Microsoft Access, SQL Server, Oracle, and FoxPro).



Overview of JDBC Technology

- JDBC provides a standard library for accessing relational databases
- API standardizes
  - way to establish connection to database
  - approach to initiating queries
  - method to create stored (parameterized) queries
  - the data structure of query result (table)
    - Determining the number of columns
    - Looking up metadata, etc.
- API does not standardize SQL syntax
- JDBC is not embedded SQL
  - JDBC classes are in the java.sql package

Functions of JDBC

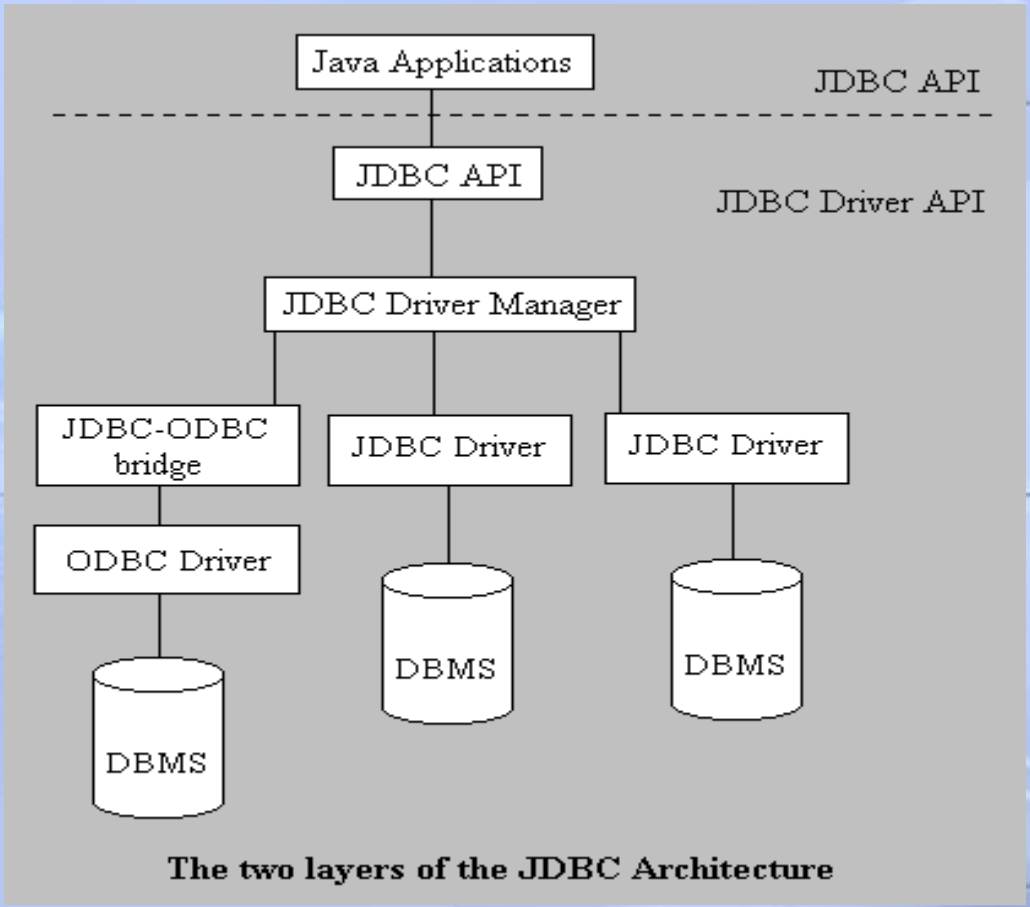
- Using JDBC, it is easy to send SQL statements to virtually any relational database.

- JDBC makes it possible to do three things:
  - establish a connection with a database
  - send SQL statements
  - process the results

On-line Resources

- Sun's JDBC Site
  - **http://java.sun.com/products/jdbc/**
- JDBC Tutorial
  - **http://java.sun.com/docs/books/tutorial/jdbc/**
- List of Available JDBC Drivers
  - **http://industry.java.sun.com/products/jdbc/drivers/**
- API for java.sql
  - **http://java.sun.com/j2se/1.4/docs/api/java/sql/package-summary.html**

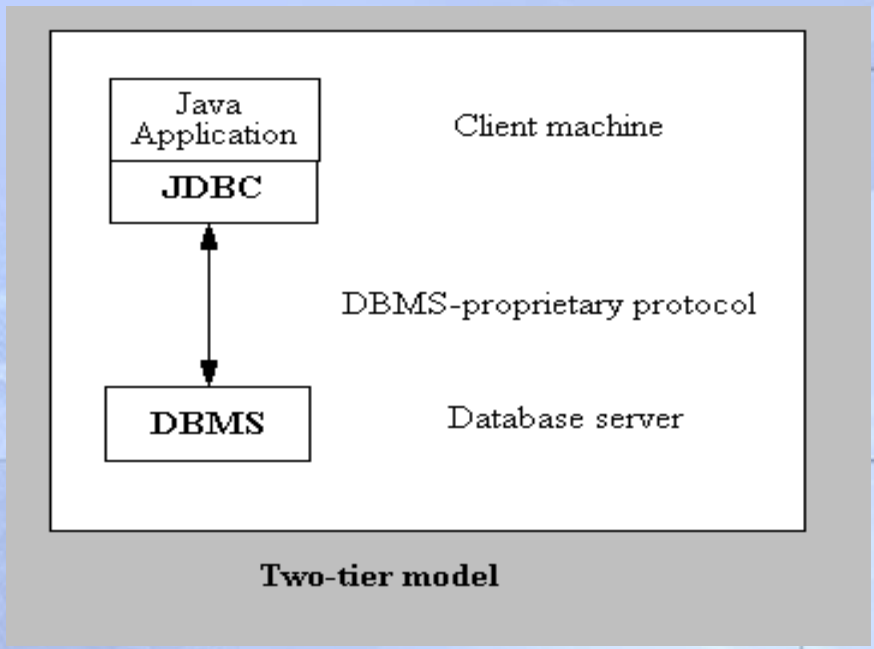
JDBC Architecture



The two layers of the JDBC Architecture

- consists of two layers:
  - the JDBC API, which provides the application-to-JDBC Manager connection, and
  - the JDBC Driver API, which supports the JDBC Manager-to-Driver Connection.

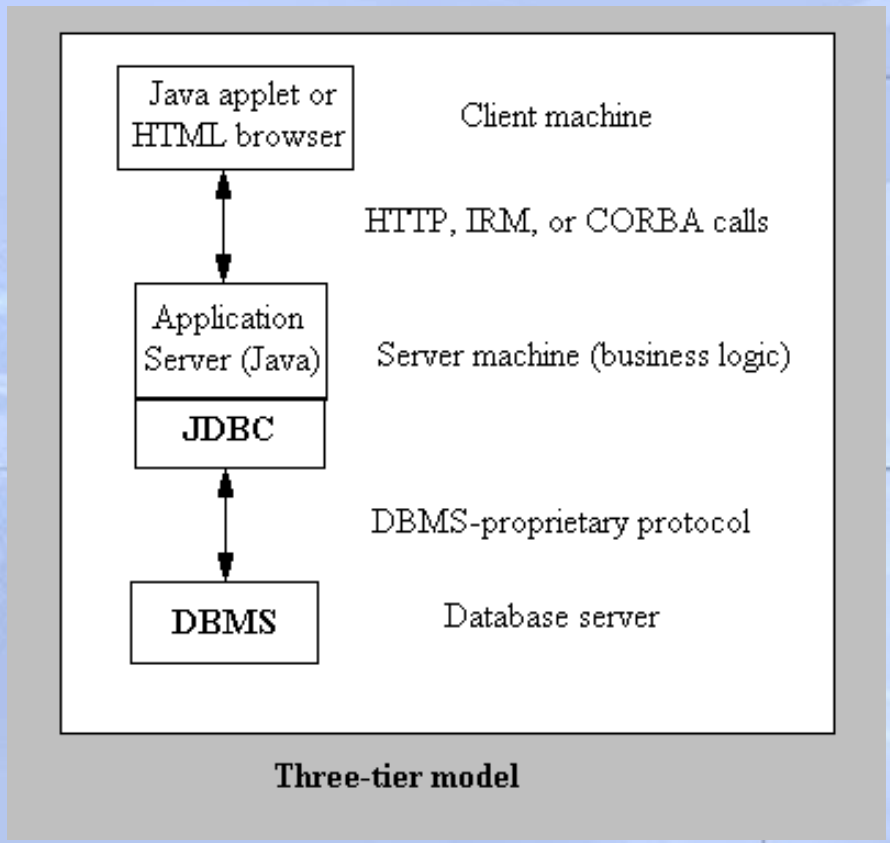
Two-Tier Model Supported by JDBC



Two-tier model

- Java applet or application talks directly to the database
- referred to as a client/server configuration

Three-Tier Model Supported by JDBC



Three-tier model

- possible to maintain control over access and the kinds of updates that can be made to corporate data
- the middle-tier architecture can provide performance advantages

JDBC API

- The Java environment provides you the JDBC API necessary to create Java applications that are capable of interacting with a database.

- It is a simple class hierarchy for database objects.

- Some classes contained in the **java.sql** package.
  - **java.sql.DriverManager** : loads driver, and creates the connection to the database
  - **java.sql.Driver** : represents a driver
  - **java.sql.Connection** : represents a connection to the database
  - **Java.sql.Statement** : executes statements
  - **Java.sql.ResultSet** : this holds results of executing statement

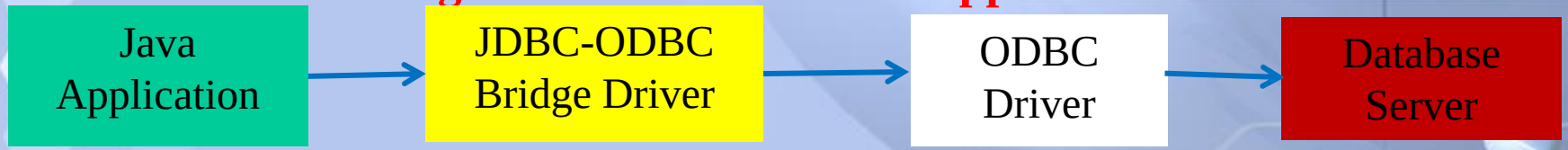
JDBC Drivers Types

- JDBC drivers fit into one of the following four categories:
  - **Type 1 : The JDBC-ODBC Bridge plus ODBC driver**
  - **Type 2 : Native-API partly-Java driver**
  - **Type 3: JDBC-Net pure Java driver**
  - **Type 4 : Native-protocol pure Java driver**

Type 1 : The JDBC-ODBC Bridge plus ODBC driver

- This is the technology that lets Java Applications use existing ODBC drivers.
- The bridge driver translates JDBC method calls into ODBC function calls and sends them to ODBC Driver.
- The ODBC Driver then forwards the call to database server.
- **Advantages**
  - it allows access to almost any database
  - useful in scenarios where ODBC drivers are already installed on the client computers.
- **Disadvantage**
  - the performance is slow

✖ **Not suitable for large-scale database-based applications.**



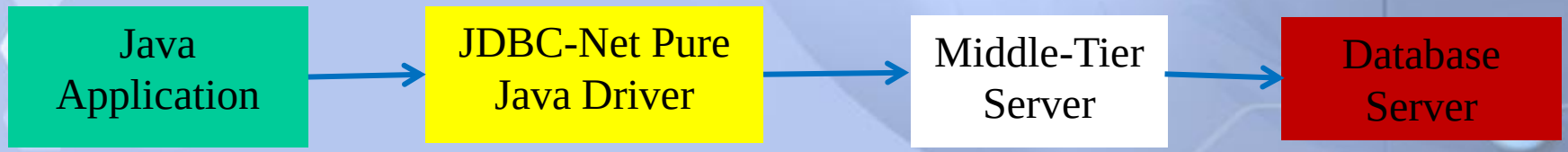
Type 2 : Native-API partly-Java driver

- This kind of driver is a **two-tier driver** that converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, or other DBMS.
- This driver connects the client to the DBMS by way of the vendor-supplied libraries for the DBMS.
- **Advantage**:
  - provide better performance
- **Disadvantage**:
  - vendor database library needs to be loaded on each client computer.

✘ **Not suitable for distributed applications.**

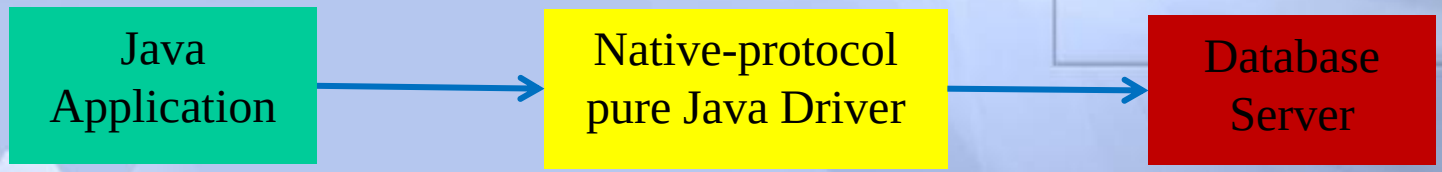
Type 3: JDBC-Net pure Java driver

- follows a **three-tiered** approach
- used to connect a client application or applet to a database over a TCP/IP connection
- the presence of any vendor database library on client computers is not required because the Type 3 JDBC driver is server-based
- **Advantage**:
  - achieve optimized portability, performance, and scalability
  - support for features such as caching of connections, query results, load balancing, and advanced system administration such as logging and auditing
- **Disadvantage**:
  - that it requires database-specific coding to be done in the middle tier
  - traversing the resultset may take longer because the data comes through the backend server.

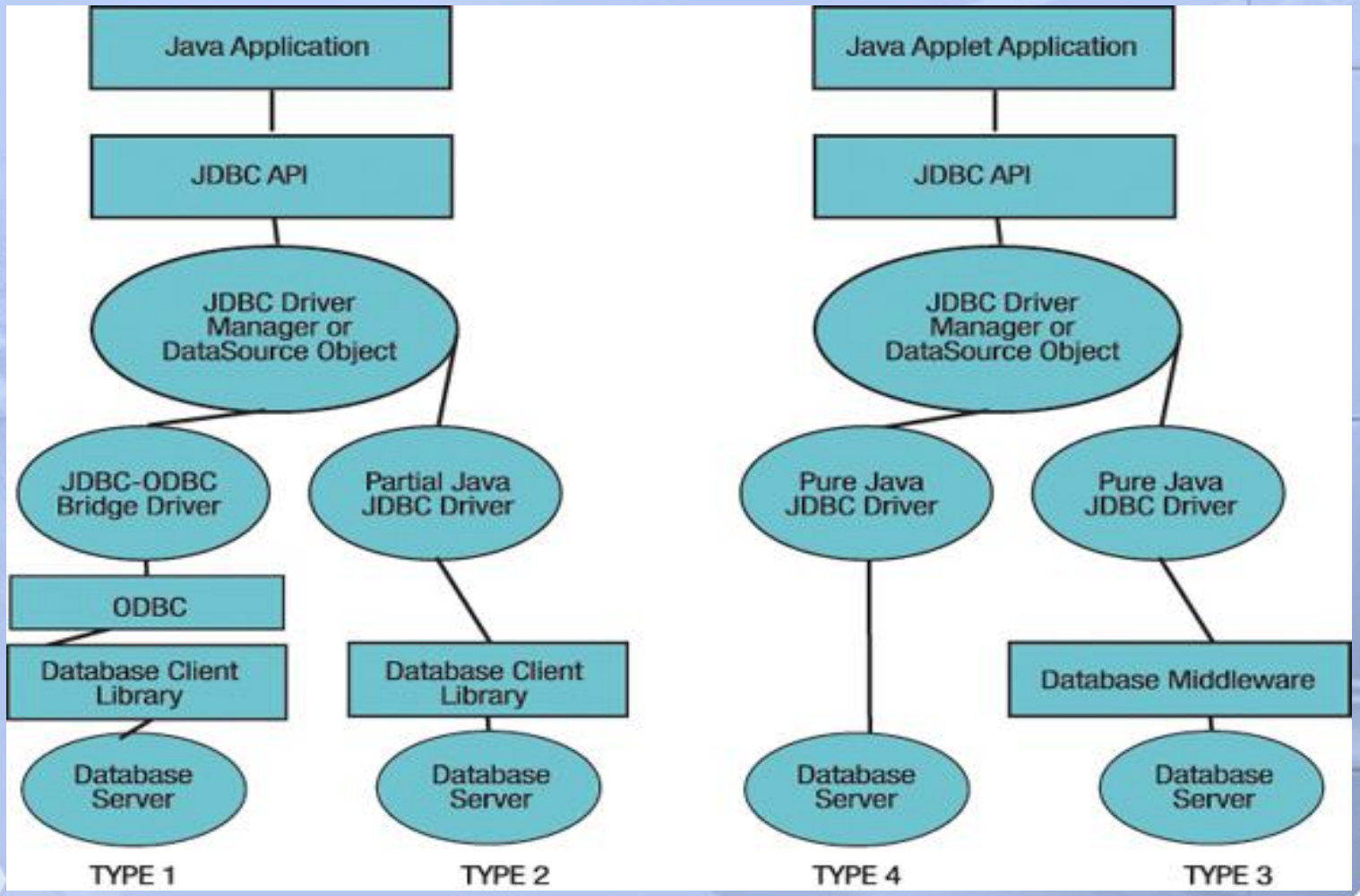


Type 4 : Native-protocol pure Java driver

- completely implemented in Java to achieve platform independence
- client applications can communicate directly with the database server
- **Advantage**:
  - do not have to translate database requests to ODBC calls or a native connectivity interface or pass the request on to another server
- **Disadvantage**:
  - a different driver is needed for each database



Comparison


Java Application
JDBC API
JDBC Driver Manager or DataSource Object
JDBC-ODBC Bridge Driver
Partial Java JDBC Driver
ODBC
Database Client Library
Database Client Library
Database Server
Database Server
TYPE 1
TYPE 2
Java Applet Application
JDBC API
JDBC Driver Manager or DataSource Object
Pure Java JDBC Driver
Pure Java JDBC Driver
Database Middleware
Database Server
Database Server
TYPE 4
TYPE 3

Java.sql Package

- The classes in the java.sql package can be divided into the following groups:
  - ✓ **Connection Management**
  - ✓ **Database Access**
  - ✓ **Data Types**
  - ✓ **Database Metadata**
  - ✓ **Exception and Warnings**

Connection Management

Java.sql.DriverManager	The basic service for managing a set of JDBC drivers.
Java.sql.Driver	The interface that every driver class must implement.
Java.sql.DriverPropertyInfo	Driver properties for making a connection.
Java.sql.Connection	A connection (session) with a specific database.

Database Access

Java.sql.Statement	The object used for executing a static SQL statement and returning the results it produces.
Java.sql.PreparedStatement	An object that represents a precompiled SQL statement.
Java.sql.CallableStatement	The interface used to execute SQL stored procedures.
Java.sql.ResultSet	A table of data representing a database result set, which is usually generated by executing a statement that queries the database.

Data Types

JDBC data types	Java Types
BIT	boolean
TINYINT	byte
SMALLINT	Short
INTEGER	int
BIGINT	long
REAL	float
FLOAT DOUBLE	double
BINARY VARBINARY LONGVARBINARY	byte[]
CHAR VARCHAR LONGVARCHAR	String

JDBC data types	Java Types
NUMERIC DECIMAL	BigDecimal
DATE	java.sql.Date
TIME TIMESTAMP	java.sql.Timestamp
CLOB	java.sql.Clob
BLOB	java.sql.Blob
ARRAY	java.sql.Array
DISTINCT	Mapping of underlying type
STRUCT	java.sql.Struct
REF	java.sql.Ref
JAVA_OBJECT	Underlying java classes

Database Metadata

Java.sql.DatabaseMetadata	Comprehensive information about the database as a whole.
Java.sql.ResultSetMetaData	An object that can be used to get information about the types and properties of the columns in a ResultSet object.
Java.sql.ParameterMetadata	An object that can be used to get information about the types and properties of the parameters in a PreparedStatement object.

Exceptions and Warnings

Java.sql.SQLException	An exception that provides information on a database access error or other errors.
Java.sql.SQLWarning	An exception that provides information on database access warnings.
Java.sql.BatchUpdateException	An exception thrown when an error occurs during a batch update operation.
Java.sql.DataTruncation	An exception that reports a DataTruncation warning (on reads) or throws a DataTruncation exception (on writes) when JDBC unexpectedly truncates a data value.

Seven Basic Steps in Using JDBC

1. Load the driver
2. Define the Connection URL
3. Establish the Connection
4. Create a Statement object
5. Execute a query
6. Process the results
7. Close the connection

JDBC: Details of Process

1. Load the driver

```
try {
        Class.forName("connect.microsoft.MicrosoftDriver");
        Class.forName("oracle.jdbc.driver.OracleDriver");
        } catch (ClassNotFoundException  cnfe) {
        System.out.println("Error loading driver: " cnfe);
        }
```

2. Define the Connection URL

URL = protocol + type of DB + host name + port + DB name;

e.g.,

```
String  accessURL="jdbc:odbc:dsn";
String oracleURL = "jdbc:oracle:thin:@" + host +":" + port + ":" + dbName;
String sybaseURL = "jdbc:sybase:Tds:" + host +":" + port + ":" +
    "?SERVICENAME=" + dbName;
```

JDBC: Details of Process (Cont.)

3. Establish the Connection

```
String username = "*****";
String password = "***";
Connection connection =DriverManager.getConnection(URL,username,password);
```

- Optionally, look up information about the database

```
DatabaseMetaData dbMetaData =connection.getMetaData();
String productName =dbMetaData.getDatabaseProductName();
System.out.println("Database: " + productName);
String productVersion =dbMetaData.getDatabaseProductVersion();
```

JDBC: Details of Process (Cont.)

4. Create a Statement

```
Statement statement =connection.createStatement();
```

5. Execute a Query

```
String query = "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet = statement.executeQuery(query);
```

– To modify the database, use executeUpdate, supplying a string that uses UPDATE, INSERT, or DELETE – Use setQueryTimeout to specify a maximum delay to wait for results

JDBC: Details of Process (Cont.)

6. Process the Result

```
while(resultSet.next())
{
        System.out.println(resultSet.getString(1) + " " + resultSet.getString(2) + " "
        + resultSet.getString(3));
} // First column has index 1, not 0
```

– ResultSet provides various getXxx methods that take a colu index or column name and returns the data

– You can also access result meta data (column names, etc.)

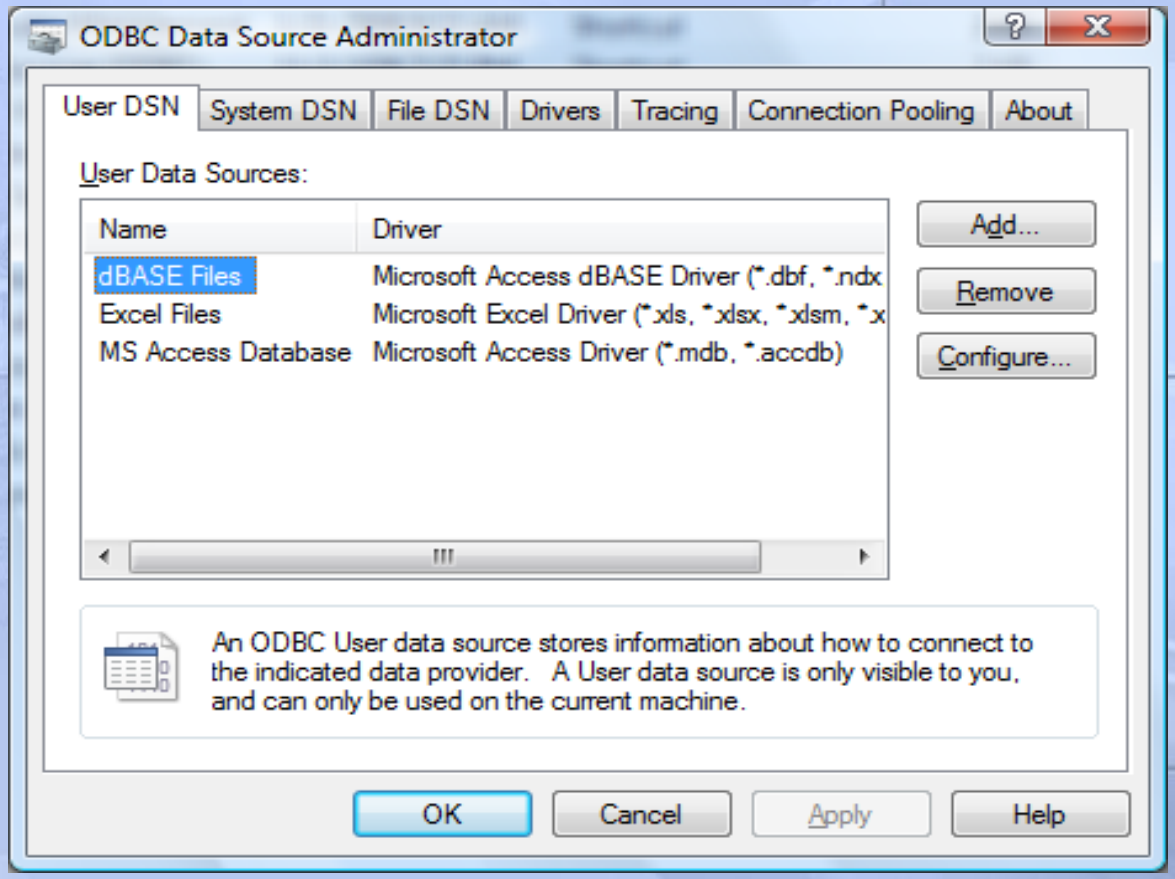
7. Close the Connection

```
connection.close();
```

– Since opening a connection is expensive, postpone this step if additional databaseoperations are expected

Example : Using Microsoft Access via ODBC

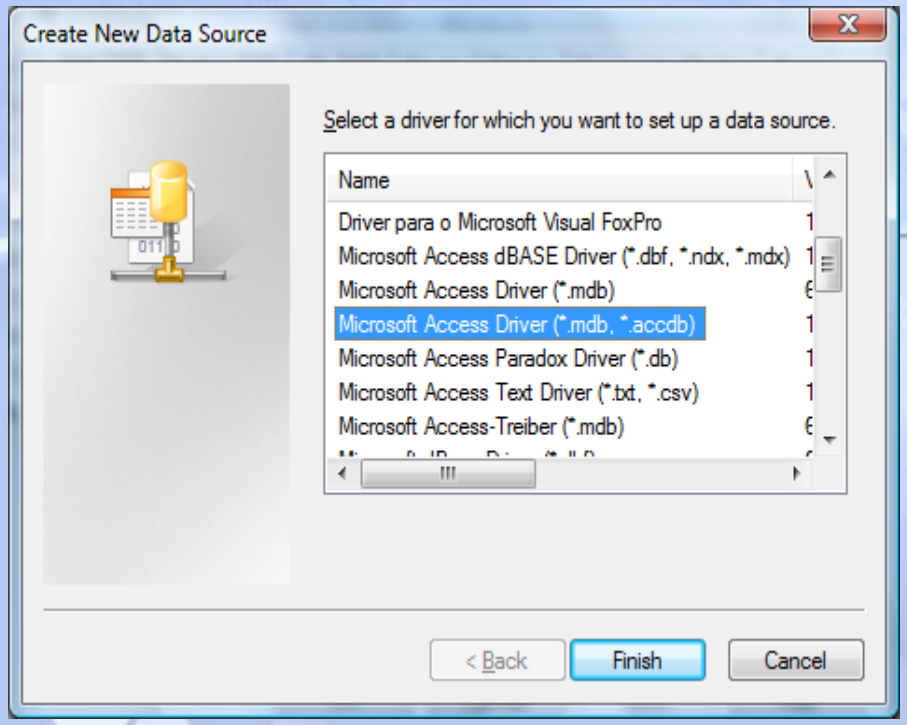
Create a Database (Database1) with a table named Student (Name, RollNo).

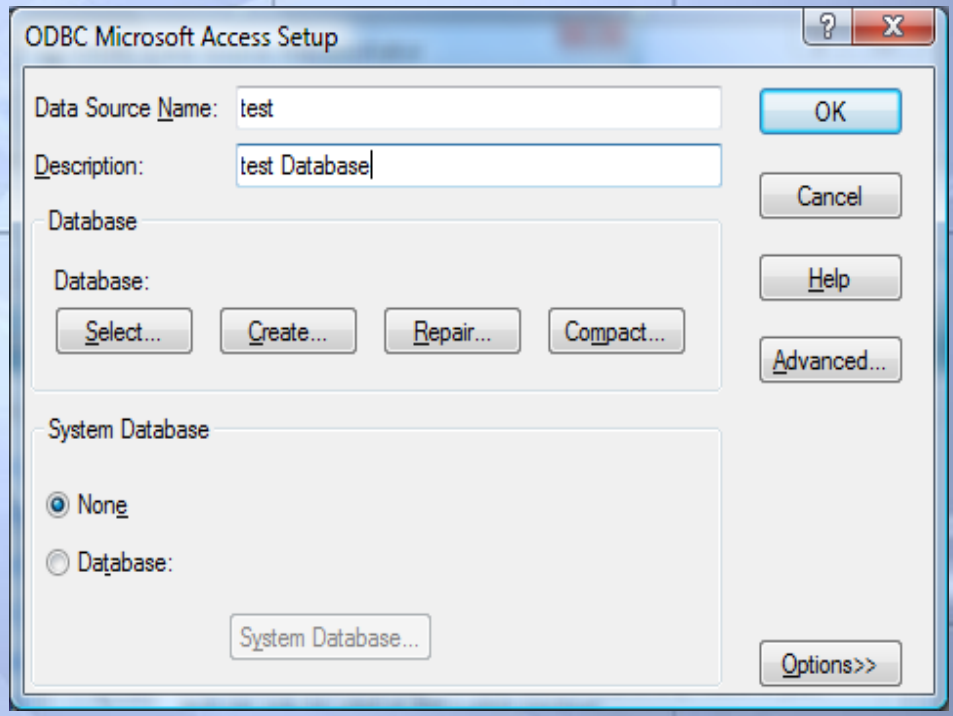


Click Start, Control Panel, Administrative Tools, Data Sources, and select Add

Example (Cont.)

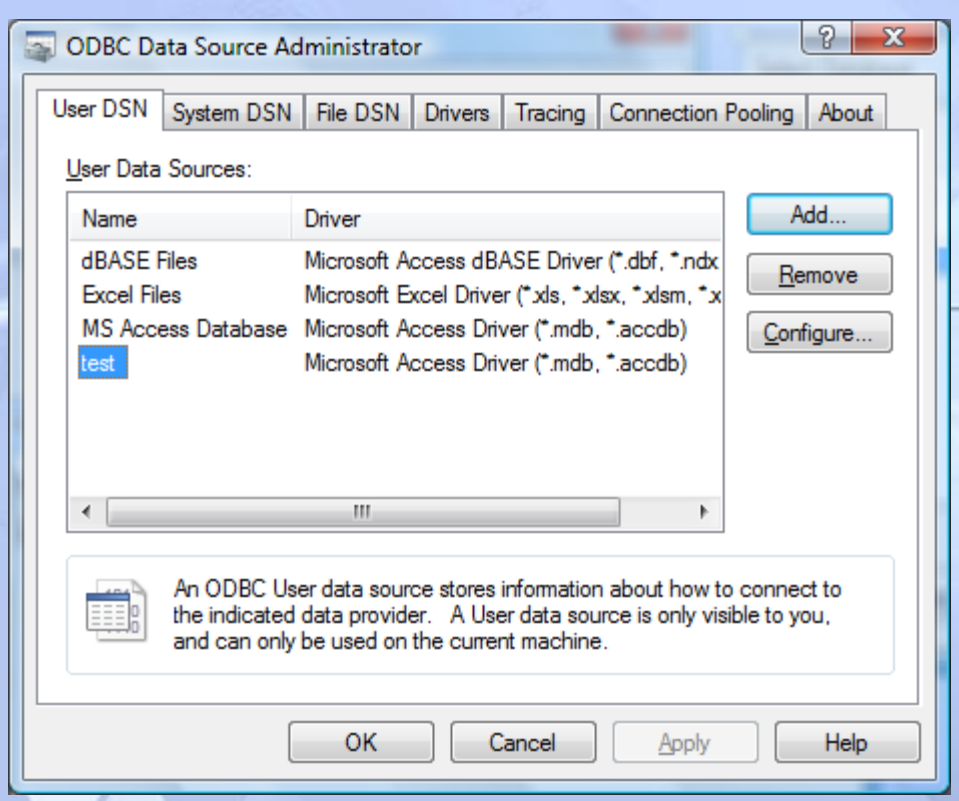
Select Microsoft Access Driver, Finish, type a name under Data Source Name, and hit Select

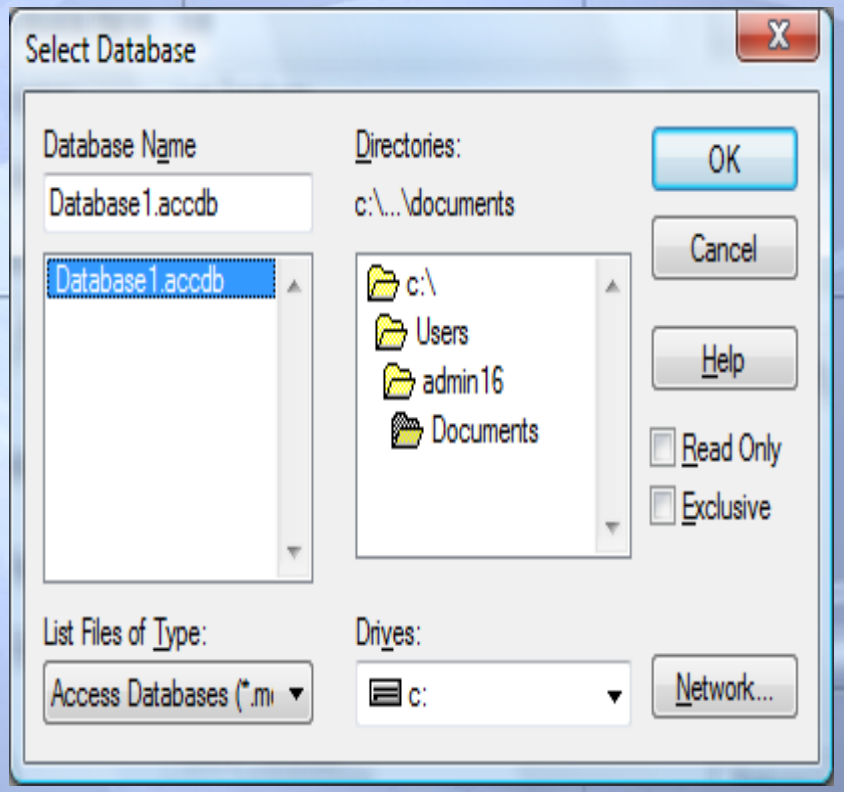




Example (Cont.)

Navigate to the directory of your database, select database, hit OK, then hit OK in following two windows.





Example (Cont.)

Use sun.jdbc.odbc.JdbcOdbcDriver as the class name of the JDBC driver.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

• Use "jdbc:odbc:Northwind" as the database address, and use empty strings for the

username and password.

```
Connection con =
DriverManager.getConnection("jdbc:odbc:test","","");
```

Example (Cont.)

```
import java.sql.*;

public class DBTest {
    public static void main(String[] args) {
        String driver ="sun.jdbc.odbc.JdbcOdbcDriver";
        String url = "jdbc:odbc:test";
        String username = "";
        String password = "";
        showEmployeeTable(driver, url,username, password);
    }
```

Example (Cont.)

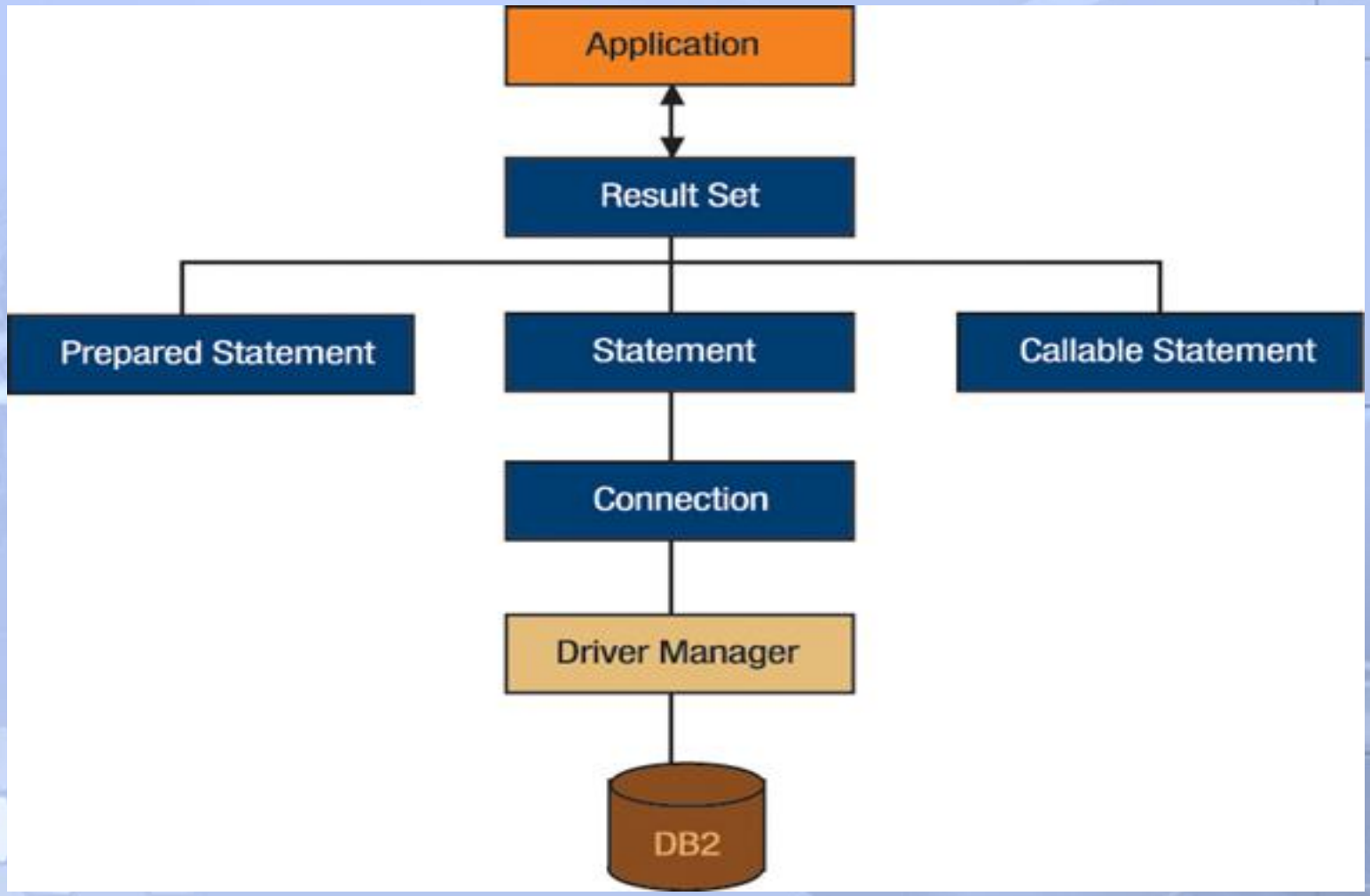
```
public static void showEmployeeTable(String driver, String url, String
    username, String password) {
 try {
    // Load database driver if not already loaded.
    Class.forName(driver);
    // Establish network connection to database.
    Connection connection =
                    DriverManager.getConnection(url,username,password);
    Statement statement =connection.createStatement();
    String query ="SELECT * FROM Student";

    // Send query to database and store results.
    ResultSet rs=statement.executeQuery(query);
```

Example (Cont.)

```
        // Print results.
        while(resultSet.next()) {
          System.out.print(resultSet.getString(1) + " "); // First column
          System.out.println(resultSet.getInt(2)); // Second column
        }
    } catch(ClassNotFoundException cnfe) {
          System.err.println("Error loading driver: " +cnfe);
    } catch(SQLException sqle) {
          System.err.println("Error connecting: " + sqle);
    }
  }//end function
}//end class
```

Three Types of Statement Classes



Using PreparedStatement

```
con = DriverManager.getConnection(url, "", "");

String sql="SELECT ID From Teachers where Name=?“;

PreparedStatement pstmt=con.prepareStatement(sql);

pstmt.setString(1,name);

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

Using MetaData

- **System-wide data**

  connection.getMetaData().getDatabaseProductName()

  connection.getMetaData().getDatabaseProductVersion()

- **Table-specific data**

  resultSet.getMetaData().getColumnCount()

- When using the result, remember that the index starts at 1, not 0

– resultSet.getMetaData().getColumnName()

Example

```
import java.sql.*;

public class DBTest {
    public static void main(String[] args) {
      String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
      String url = "jdbc:odbc:test";
      String username = "";
      String password = "";
      String tableName =“Student";
      showInfo(driver, url, username, password,tableName);
      }
```

Example (Cont.)

```
private void showInfo(String driver,String url,String username,String
password,String tableName) {
    try {
        Class.forName(driver);
    Connection connection =DriverManager.getConnection(url,
    username,password);
        DatabaseMetaData dbMetaData =connection.getMetaData();
        String productName =dbMetaData.getDatabaseProductName();
        System.out.println(“ Database: " + productName);
        String productVersion =
                    dbMetaData.getDatabaseProductVersion();
        System.out.println(“Version: " + productVersion);
```

Example (Cont.)

```
Statement statement = connection.createStatement();
String query = "SELECT * FROM " + tableName;
ResultSet resultSet = statement.executeQuery(query);
ResultSetMetaData resultsMetaData =resultSet.getMetaData();
int columnCount = resultsMetaData.getColumnCount();
for(int i=1; i<columnCount+1; i++) {
      resultsMetaData.getColumnName(i));
}
while(resultSet.next()) {
      for(int i=1; i<columnCount+1; i++) {
            System.out.print(resultSet.getString(i));
} }
```

Transactions

- The role of the transactions is to provide data integrity, correct application semantics, and a consistent view of data during concurrent access.
- Transaction Managent in the JDBC API includes the following concepts:
- Auto-commit mode
- Transaction isolation level
- Savepoint

Transaction

Idea

- By default, after each SQL statement is executed the changes are automatically ***committed*** to the database
- Turn auto-commit ***off*** to group two or more statements together into a transaction

connection.setAutoCommit(false)

- Call **commit** to permanently record the changes to the database after executing a group of statements
- Call **rollback** if an error occurs

Useful Connection Methods (for Transactions)

- **getAutoCommit/setAutoCommit**
  - By default, a connection is set to auto-commit
  - Retrieves or sets the auto-commit mode

- **commit**
  - Force all changes since the last call to commit to become permanent
  - Any database locks currently held by this Connection object are released

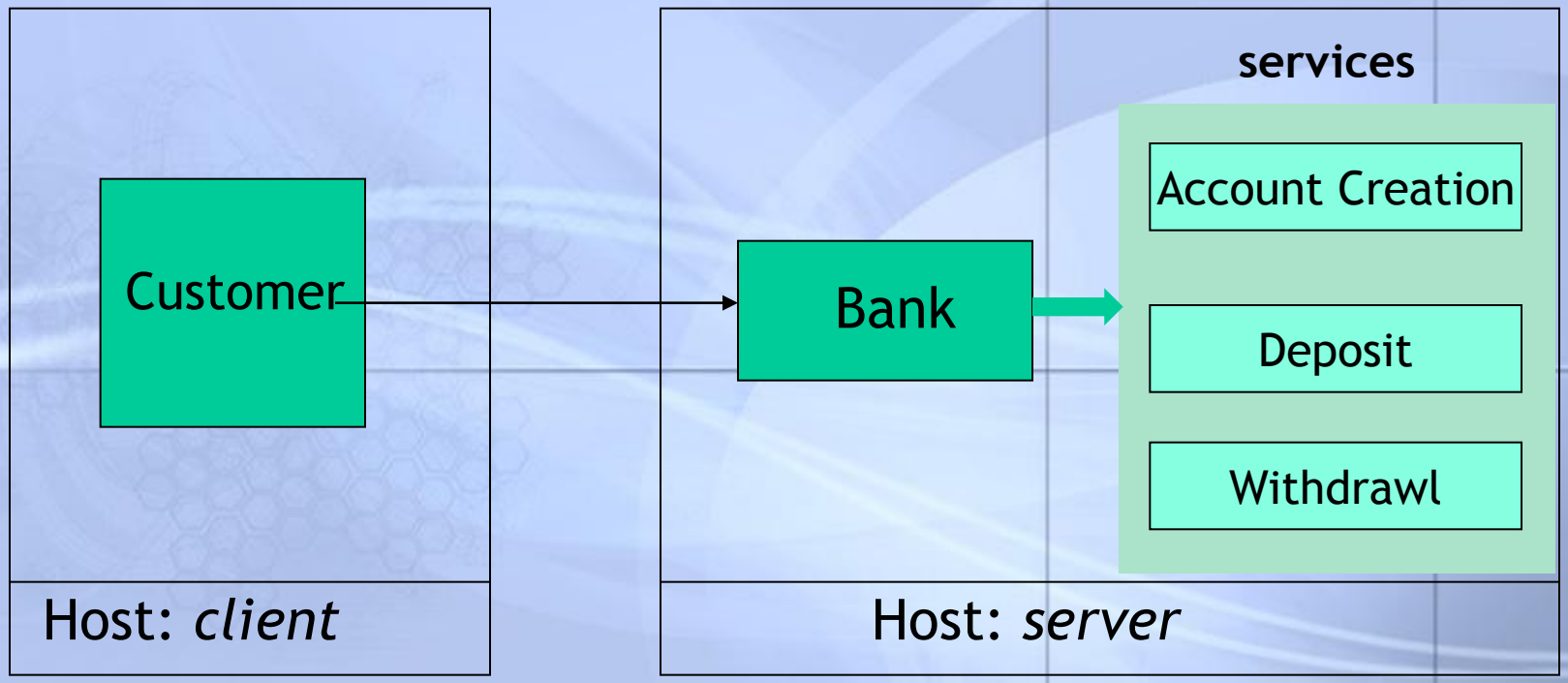
- **rollback**
  - Drops all changes since the previous call to commit
  - Releases any database locks held by this Connection object

Transaction : Example

```
Connection connection=DriverManager.getConnection(url, username, passwd);
connection.setAutoCommit(false);
try {
        statement.executeUpdate(...);
        statement.executeUpdate(...);
        connection.commit();
    } catch (Exception e) {
    try {
        connection.rollback();
      } catch (SQLException sqle) {// report problem}
    } finally {
        try {
            connection.close();
          } catch (SQLException sqle) { }
}
```

Lets make Exercise !

NetBank example



- The created bank accounts are stored in a database.
- Use JDBC to save the new bank account and update balance.

Thank You!