High performance. Delivered.

Skills to Succeed

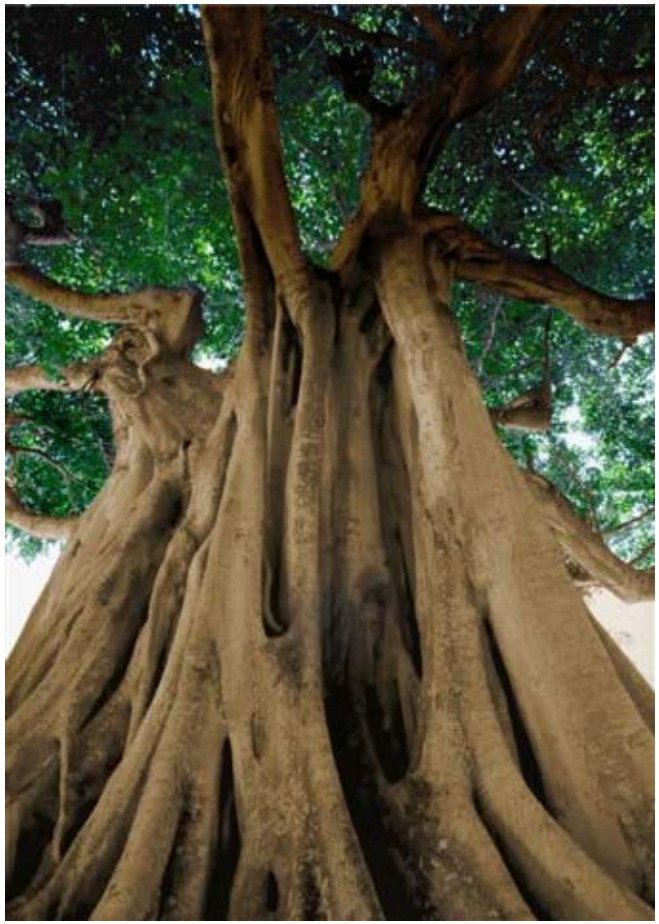
JDBC Introduction

accenture

consulting technology | outsourcing

Accenture Overview

- Global management consulting, technology services and outsourcing company
- More than 246,000 people serving clients in over 120 countries
- Clients, spanning the full range of industries – 92 of the Fortune Global 100 and more than three-quarters of the Fortune Global 500
- www.accenture.com



Accenture in Romania

- Accenture has operated in Romania since 2003 and is continuously consolidating its presence in all three pillars:
  - Management Consulting
  - Technology
  - Business Process Outsourcing (BPO)

- Romanian office:
  - Bucharest, West Gate center

Who am I?

Edis Ali

Analyst programmer

Specialty: java, sql, OSS(Service Delivery, Service Activation)

Personal interest: I like Italian and Turkish cuisine, to watch sci-fi and fantasy movies, anime, computer games

Introduction to JDBC
Course Map

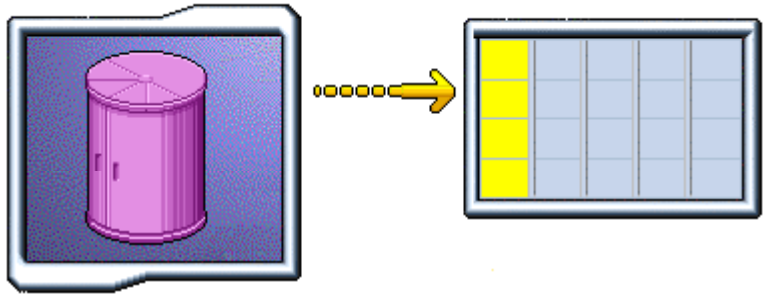
- JDBC architecture and driver concepts
- Using JDBC API
- Result set and metadata
- Procedure statements and stored procedures
- Managing transactions
- Sample application that connects to a Database

JDBC architecture and driver concepts

- JDBC architecture and driver concepts:
  - JDBC Concepts features
  - JDBC architecture
  - Selecting a JDBC driver

Relational databases

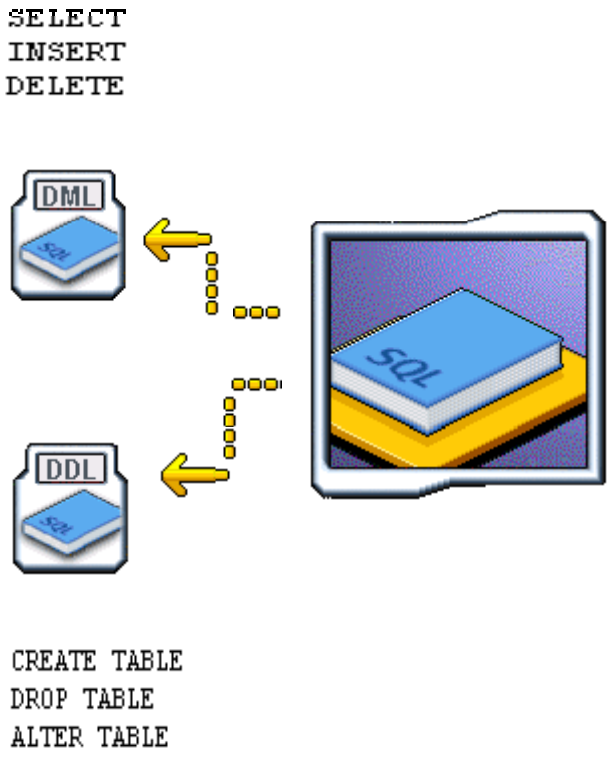
- are the most common means of storing information.
- stores information's in a tabular format, in rows and columns.



- What is a Database Management System (DBMS) ?
- What is Structured Query Language (SQL) ?

SQL Commands

- SQL Commands are divided into two categories:
  - Data Manipulation Language(DML)
  - Data Definition Language(DLL)

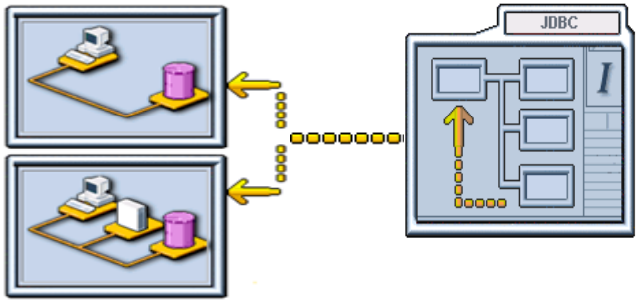


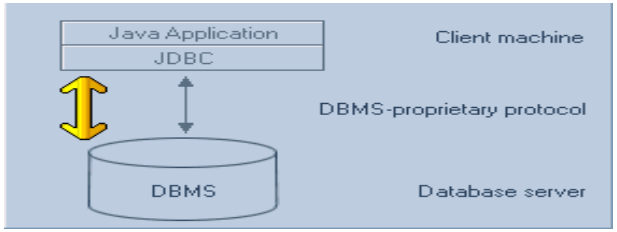
Java Database Connectivity(JDBC) (1/3)

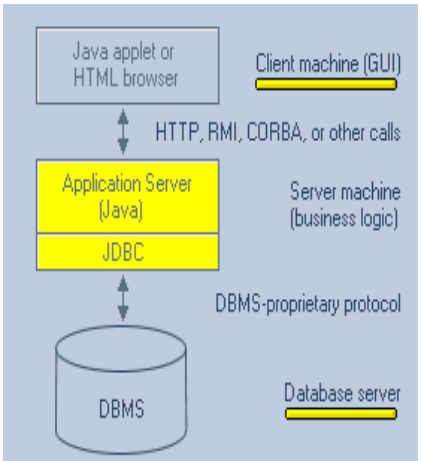
- What is Java Database Connectivity(JDBC) ?
- Java Database Connectivity(JDBC):
  - enables to send SQL statements to any type of relational database from JAVA application.
  - provides classes and interfaces used to write databases applications in 100% Pure Java.
  - is secure and portable across any kind of Java Platform.
  - JDBC API classes
  - API

Java Database Connectivity(JDBC) (2/3)

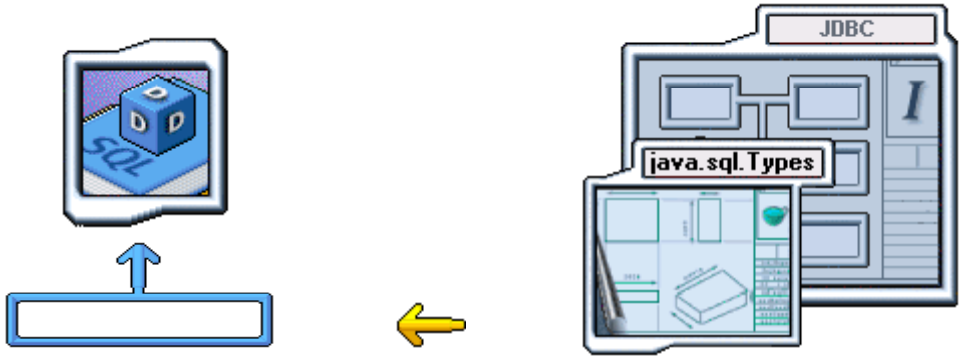
- JDBC provides support for both **two-tier** and **three-tier** database access models:
  - In a **two-tier** model Java application model has direct communication with the database
  - In a **three-tier or multi-tier** model a middle tier processes communication between a client application and a database:







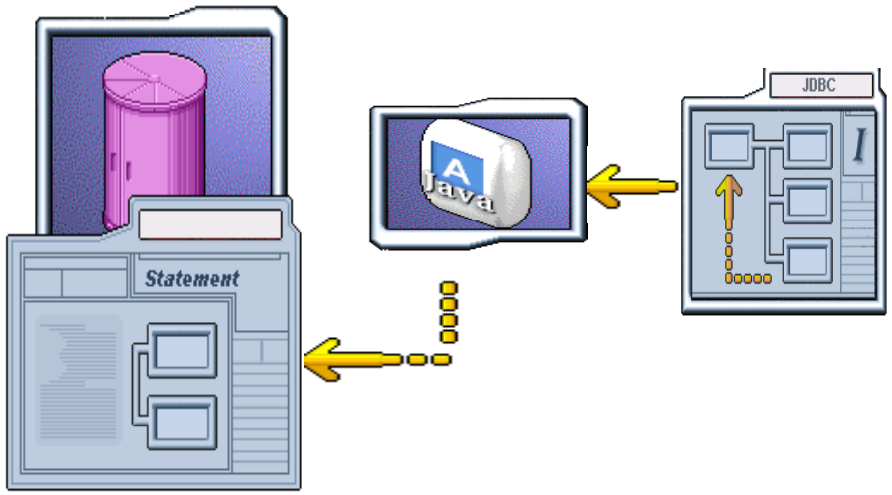
Java Database Connectivity(JDBC) (3/3)



- JDBC defines a general set of SQL type identifiers provided by the java.sql.Types class in order to solve the problem of the different SQL datatypes of the DBMSs
- JDBC API
- JDBC escape clauses

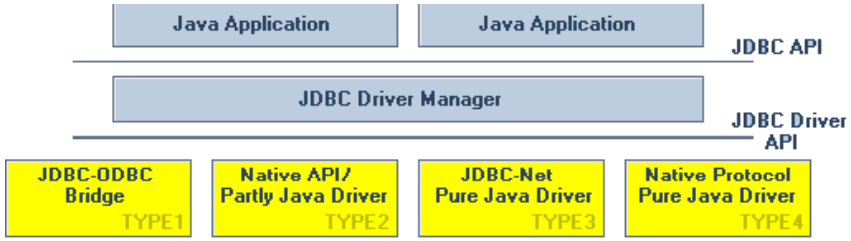
JDBC API (1/2)

- JDBC API allows to:
  - connect to a database
  - execute SQL Statements
  - process the result of the SQL statements



JDBC API (2/2)

- The JDBC API can be divided into two parts:
  - an API at user level that processes SQL statements and manages JDBC Driver Manager communication
  - a driver API

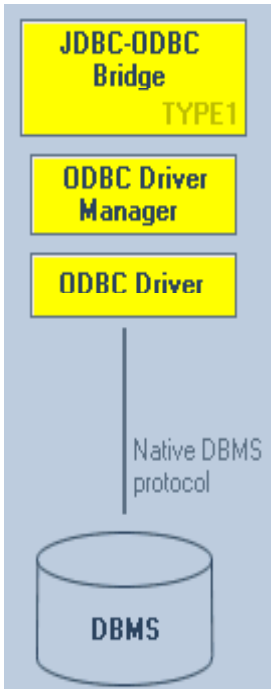


- The JDBC Driver Manager connects your Java application with the appropriate JDBC driver.

JDBC drivers(1/5)

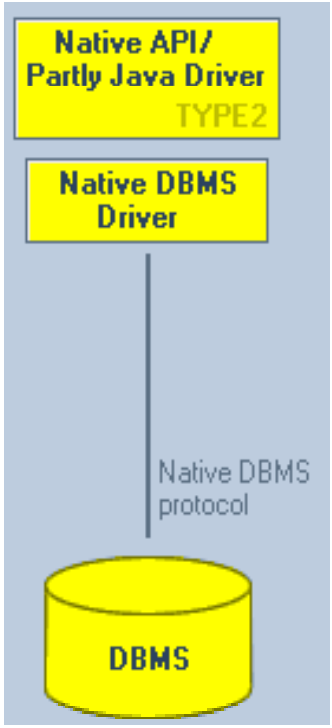
- JDBC drivers can be categorized into four types:
  - Type 1: JDBC-ODBC bridge plus ODBC driver
  - Type 2: Native-API partly-Java driver
  - Type 3: JDBC-Net Pure Java driver
  - Type 4: Native-protocol Pure Java driver

JDBC-ODBC bridge plus ODBC driver(2/5)



- Converts JDBC calls into Microsoft's Open Database Connectivity (ODBC)
- ODBC is a programming interface
- ODBC cannot be accessed from Java without using an ODBC-JDBC bridge.

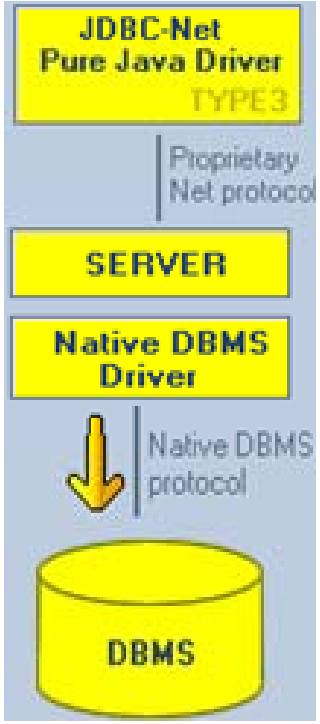
Native-API partly-Java driver(3/5)



- Converts JDBC calls into calls on the native API for the target database.
- Driver binary code needs to be loaded on each of the client machines.

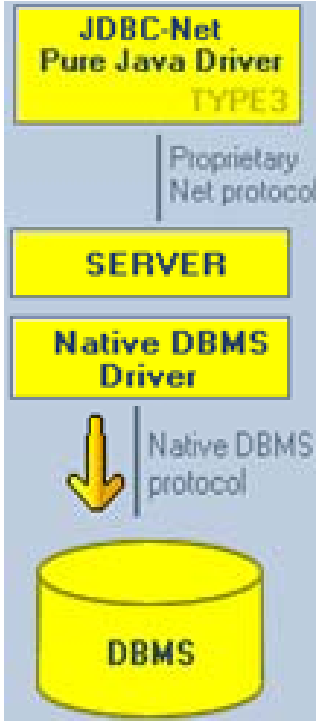
.

JDBC-Net Pure Java driver(4/5)



- Translates JDBC calls into a DBMS-Independent network protocol.
- Is a 100% Pure Java implementation code on client machines.

Native-protocol Pure Java driver (5/5)



- Translates JDBC calls into the API that's used by the target database, making it highly database-specific.
- The client machine can access the database server directly.
- Is a Pure Java implementation

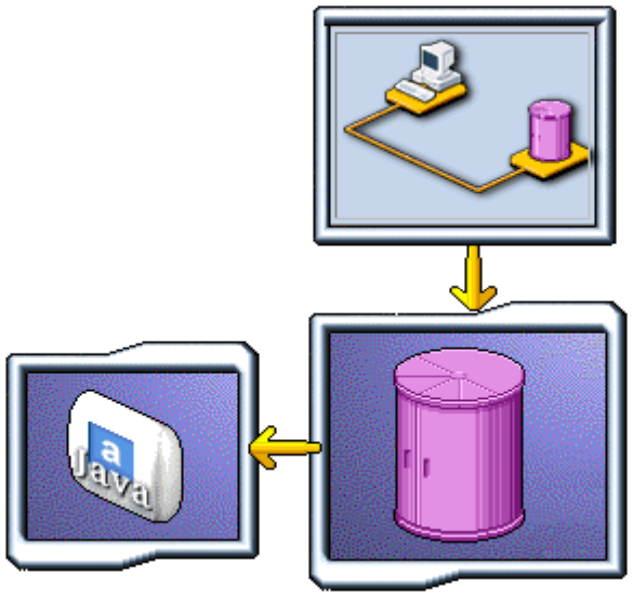
JDBC driver vs. Type of applications(1/4)

- The JDBC driver is based on the design of your application and its data access requirements.

- Three types of application architecture to consider when selecting an appropriate driver:
  - two-tier architecture: the database clients are deployed as applets.
  - two-tier architecture: the database client is deployed as a regular Java application.
  - three-tier architecture: the database client can be deployed as an applet or a Java application.

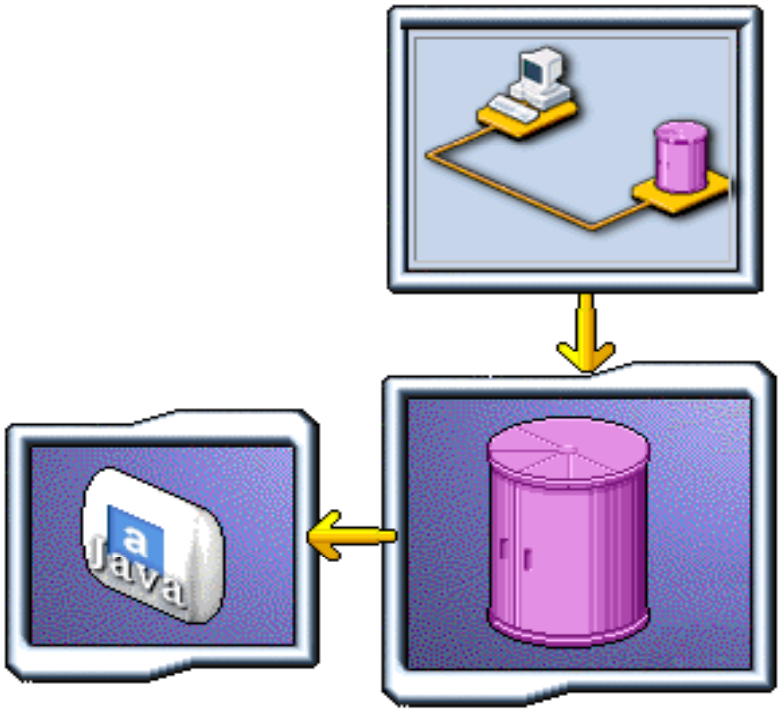
Two-tier - database clients are deployed as applets(2/4)

- Driver Type 4 - the most suitable one



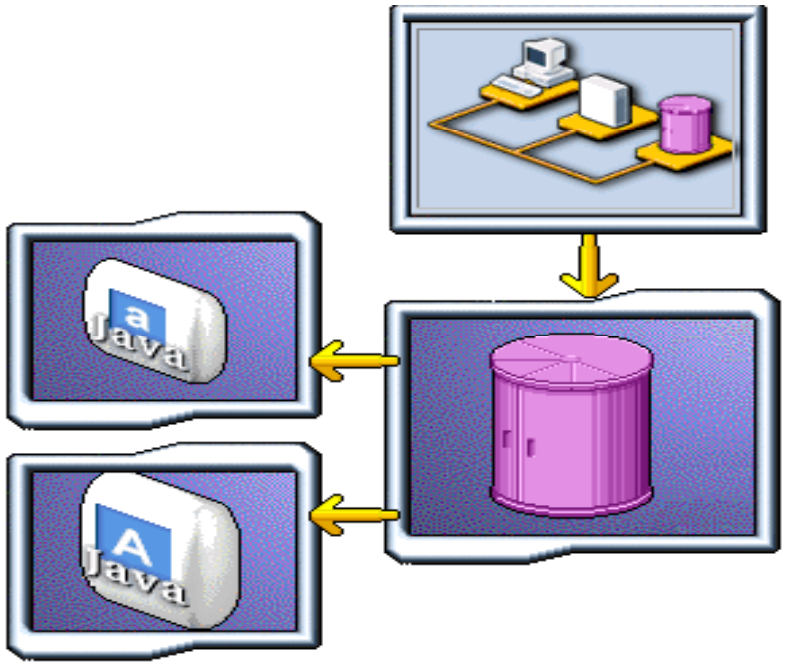
Two-tier - database client is deployed as a regular Java application(3/4)

- Driver Types 1,2 or 4 - the most suitable



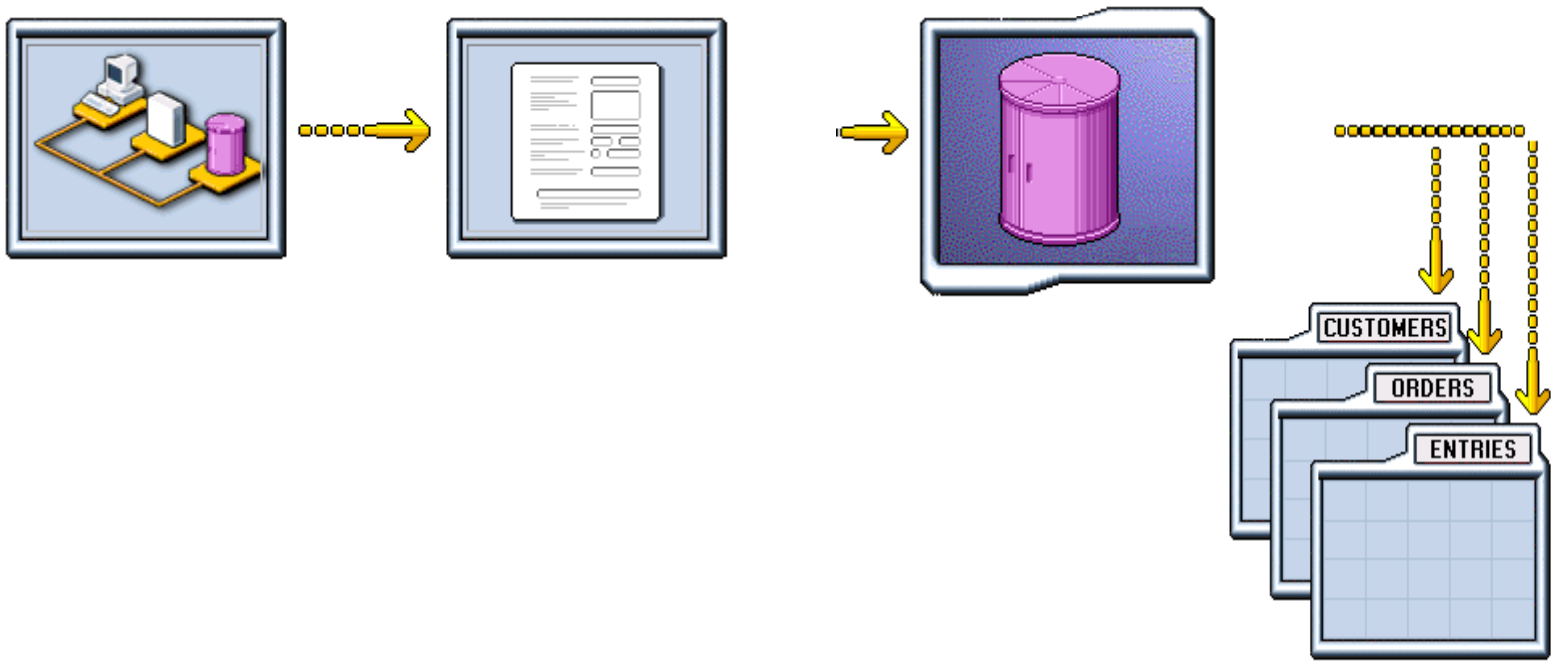
Three-tier - database client can be deployed as an applet or a Java application(4/4)

- Any Type of Driver can be used, but Driver Types 2, 3 and 4 are more appropriate

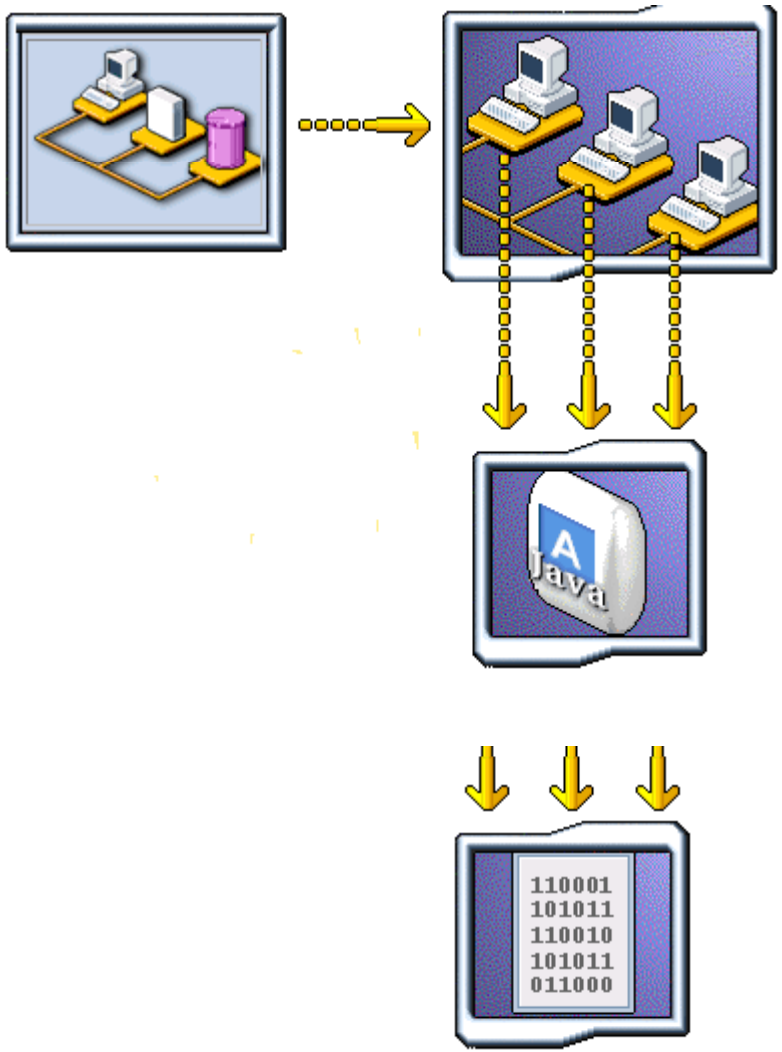


Example of multi-tier database connecting (1/3)

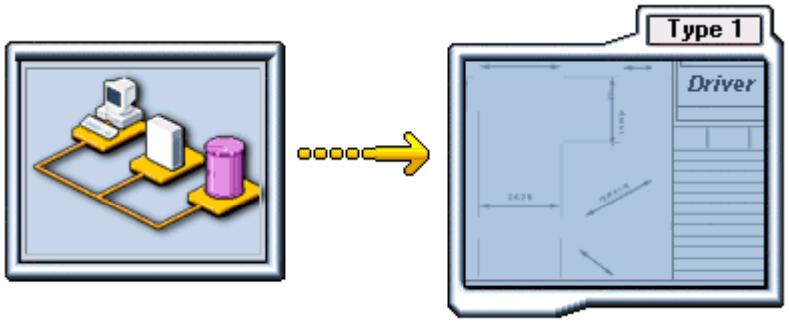
- Let's say that you want to develop a multi-tier order entry system using all the Java enterprise technologies.



Example of multi-tier database connecting (2/3)



- The type 1 driver can be suitable for use in this architecture.



Example of multi-tier database connecting (3/3)

- When connecting to a database using JDBC, you need to specify a JDBC URL:

```
jdbc.subprotocol:subname
```

- The URL provides your application with the information it needs to:
  - choose a driver
  - identify the data source
- In our example the URL looks like this:

```
jdbc:odbc:data-source-name
```

Using JDBC API

- Using JDBC API:

– The JDBC API
– Issuing databases requests

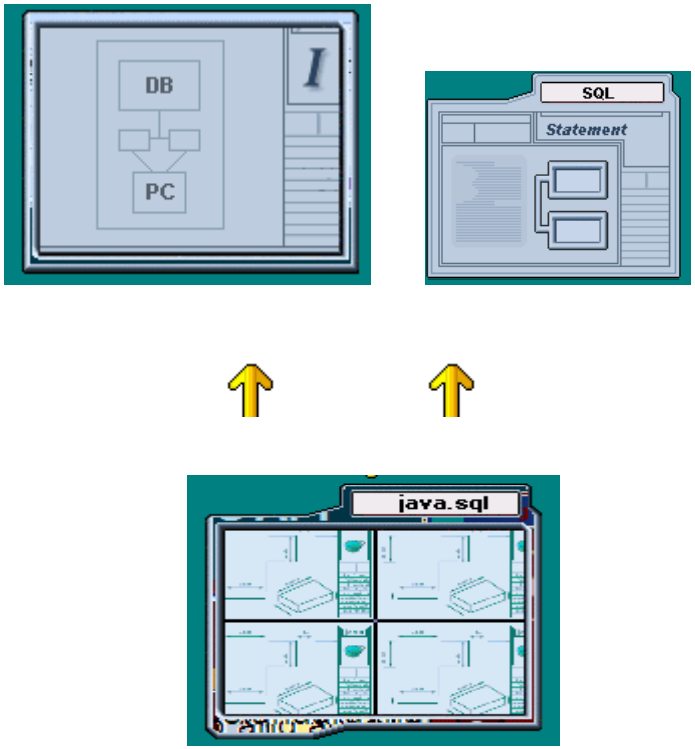
Using JDBC API (1/2)

- Java Database Connectivity Application Programming Interface (API) is an industry standard that provides universal data access.
- JDBC API is implemented by the java.sql package.
- Java.sql package provides interfaces that allow you to:

– connect to and access a database

– issue SQL statements

– process the results



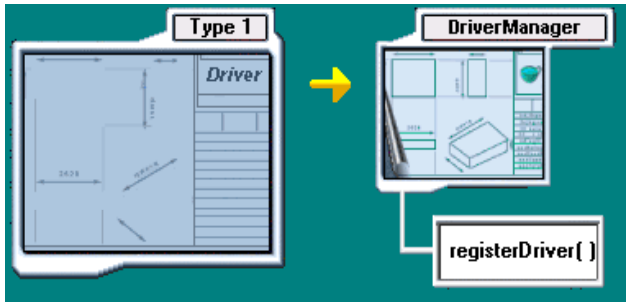
Using JDBC API (2/2)

- The `java.sql.Driver` interface handles the way in which drivers are loaded and supports the creation of new database drivers.
- The `java.sql.Connection` interface provides the functionality to make a connection to a database.
- The `java.sql.Statement` interface handles the way in which SQL statements are issued against that connection.
- The `java.sql.ResultSet` manages access to the data rows returned by a SQL SELECT statement.

Loading JDBC driver(1/3)

- Before making a connection to a data source you need to specify and load a suitable JDBC driver which needs to implement the java.sql.Driver interface.
- This URL-like syntax is used to specify an appropriate driver class name:

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



- When a driver class is loaded, it must register itself with the **DriverManage**r using the registerDriver method.

Loading JDBC driver(2/3)

- Once the driver registers itself at load time, you can load it by invoking the static **Class.forName method**:

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

- This driver - `sun.jdbc.odbc.JdbcOdbcDriver` - is the standard type 1 driver delivered with the **Java Developer Toolkit (JDK)**.

- The **DriverManager class** incorporates a number of static methods that allow you to specify connection time and monitor logging information associated with your driver.

Driver manager Methods

- The `setLoginTimeout method` is used to specify the maximum time in seconds that drivers have to wait when connecting to a database.

```
public static void setLoginTimeout(int seconds)
```

- The `setLogStream method` allows you to trace and log DriverManager output information and store it in a log file.

```
public static void setLogStream(Printstream out)
```

Establish a database connection(1/2)

- To establish a database connection, you need to obtain a Connection object by calling the `getConnection static method` of the DriverManager class.

```
Connection con=DriverManager.getConnection(String URL, String user,
String password)
```

- `getConnection method` contains:
  - ”URL”: information about the data source location
  - user, password: host database username and password

Establish a database connection(2/2)

- The string URL that you provide depends on the driver that you are using.
- The URL is followed by the username that you use to log into the particular database that you're connecting to.

– if your database login name is Joe and your password is Je, this is how you make a connection to the database corresponding to the ODBC32 DSN called DAT:

```
String url="jdbc:odbc:DAT";
String user="Joe";
String password = "Je";
Connection con=DriverManager.GetConnection(URL,user,password);
```

Issuing databases requests(1/4)

- Establish the connection with the database.
- Create a Statement object from the connection,which implements the Standard interface.

```
Statment =
  con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR UPDATABLE);
```

- The SQL code is passed into the Connection object, which forwards the information to the appropriate driver.

Issuing databases requests(2/4)

- The Connection object's `createStatement` method returns a statement object.
- The Statement interface provides three methods that are used to send different types of SQL statement to a database:

– `executeQuerry()`:used to obtain query information from a database(SELECT)

– `executeUpdate()`: used to update the values in a database (INSER,UPDATE,DELETE, CREATE)

– `execute()`: can be used to issue any statements to the database

Issuing databases requests(3/4)

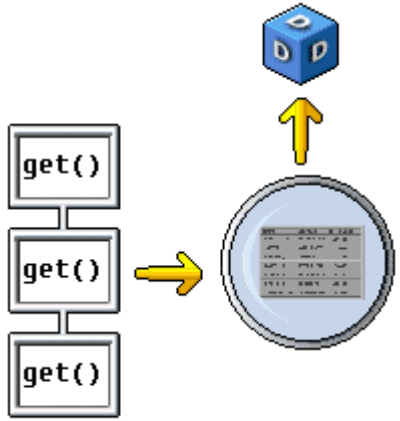
- `Statement object` called statement is created from the `Connection object, con`:

```
String url="jdbc:odbc:DAT";
String user="Joe";
String password = "Je";
Connection con=DriverManager.GetConnection(URL,user,password);
Statement statement  = con.createStatement();
```

Retrieving information and processing

- All data rows returned by the SQL query are returned through a single ResultSet object,data that can be retrieved using a get methods:

```
getObject() ;getString();   getLong();
```



- To move to the next row of the ResultSet object by invoking its next method.
- To update the values in a database, you invoke a Statement objects's executeUpdate method.

How to issue a SELECT statement to return a simple ResultSet object(1/2)

- First create a Statement object and use its executeQuery method to return a ResultSet object called results:

```
        Resultset result=statement.executeQuerry("SELECT *
FROM"+"CUTOMMERS");
```

- The ResultSet object will consist of the data returned as the result of the SELECT statement passed to the executeQuery method:

```
     Resultset result=statement.executeQuerry("SELECT *
FROM"+"CUTOMMERS");
      results.next();
      String custnum = results,getString(1);
      System.out.println("Customer Number is " + custnum);
```

How to issue a SELECT statement to return a simple ResultSet object(2/2)

- Set the result set's cursor to the first row returned by using result.next method, which moves the cursor to the next row:

```
result.next()
```

- To return column data from a row as a String object, you use the getString method, specifying the index of the column you require as an argument to getString ()

```
String custnum=resultset.getString(1);
```

- Explicitly closing the Connection and Statement objects ensures DBMS resources are free when they are no longer needed.

```
statement.close();
con.close();
```

Exceptions

- Can occur when connecting to a database and retrieving or updating its data, so the exception might be generated by JDBC itself or by Java at run time.
- An exception prevents an application from running.
- Exceptions relating to the DBMS or to JDBC itself are incorporated in the SQLException class and its subclasses and consists of:
  - a string error message that describes the error
  - a SQL state string that identifies the error according to the X/Open SQLState conventions
  - the driver vendor's error code
- How to handle exceptions ?

– Incorporating a try and catch block in your code enables you to handle these exceptions.

SQLWarnings(1/2)

- A **SQLWarning object** alerts the user about any database access and connection requests that did not complete entirely as planned.

- The **SQLWarning class** is a subclass of **SQLException**.

– JDBC can issue a warning and still execute, for example, it may issue a warning if an error occurred on a request to disconnect from a database.

- Warnings can be reported on:

– Connection objects

– Statement objects

– ResultSet objects

SQLWarnings(2/2)

- To view a warning for a particular object, you need to explicitly call its `getWarnings method`.

```
SQLWarning w =  con.getWarnings() ;
while (w ! = null)   {
     System.out.printl(w.getSQLSTate());
     w.getNextWarning();
  }
```

Result Sets and Metadata

- Result Sets and Metadata:

– Working with result sets
– Accessing metadata

Result Sets and Metadata(1/3)

- A **ResultSet object** incorporates a table containing the results of a SQL query.
- The **ResultSet object** incorporates a cursor that points to the current row of data.
- Each time you call the **ResultSet object's next method**, the pointer scrolls to the next row in that result set.

NAME	ADDRESS	I.D. NUMBER
Samual, A	Arcadia Rd.	12345678
Scott, L	Wells Ave.	12345677
Serling, R	Oriel St.	12345676
Smith, J	Irvine Tce.	12345675
Spade, H	Holden Cls.	12345674
Stone, D	Green Pk.	12345673
Summers, M	Spencer Wk.	12345672

Result Sets and Metadata(2/3)

- Initially, the **pointer is positioned outside the first row,** when the next method is called for the first time, the pointer points to the first row.
- The **results are retrieved in a result set** from the top row down to the bottom as the pointer moves sequentially through each row.
- The pointer is active until you close the **ResultSet object** or close the Statement object that created it.
- The **get methods** of the **ResultSet object** enable you to obtain data from a column of the current row.

Result Sets and Metadata(3/3)

- These values can be retrieved in any order, but it is preferable to retrieve values from left to right.

- This ensures that optimum database portability is maintained for your code.

- The columns in a **result set** are numbered **from left to right, starting at index 1**, as the figure shows.

Working with results sets – getMethod(1/3)

- To specify the column from which you are retrieving the necessary data, the **column name** or the **actual column number** can be provided:

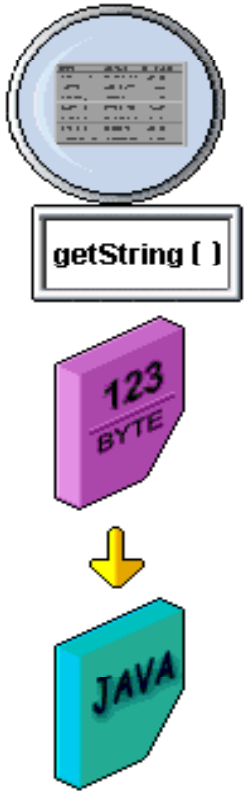
```
String s = rs.getString("cust no");

String s = rs.getString(1);
```

- As an argument to the a **get method**, the column name or column number can be assigned.

```
getString ()
```

Working with results sets - getMethod(2/3)



- By using the column name as a parameter of a **getX method,** the method returns the value from the first column.In such cases it is more appropriate to use column numbers.
- When a specific **getX method** is provided, the **JDBC driver** tries to convert the database datatype to the **Java** type that you specify. And it returns the appropriate value as a Java type.
- So X depends on the Java datatype that you require.

Working with results sets – getMethod(3/3)

- The **getInt** method should be used to retrieve INTEGER datatypes as a Java int.
- The **getString** method returns the value as a Java String object.
- The **getFloat** can be used to convert and return REAL or DECIMAL database datatypes as a Java float.
- When using the **JDBC API**, it is important to know the Java mappings for standard SQL datatypes.

Working with results sets – UpdateMethod(1/3)

- **JDBC 2.0** allows to modify a database more efficiently, by updating the rows in a result set.

- You navigate to the row in the result set that you want to update using **the first, previous, last, or next method**, then you call an **updateX method** to change the entry.

- What you put in place of X depends on the datatype of the column containing the value that's to be updated.

Working with results sets – UpdateRow(2/3)

- If you want to update the first row in a column:

```
result.first();
result.updateInt("cust_num", 3001)
result.UpdateRow();
```

Working with results sets – UpdateRow(3/3)

- The **ResultSet** object's **updateRow** commits the changes to the database.
- Before calling **updateRow (),** you can cancel any changes using **ResultSet's cancelRowUpdates method**.

Working with results sets – ADD ROW

•To add a **new row to both the result set and database** call the **ResultSet object's insertRow method.**

- JDBC 2.0 allows you to **add a new row** to a result set by calling **moveToInsertRow ()** and **insertRow ()** on the **ResultSet object**.

- To add a **row of values** to a result set,you need to create a **ResultSet object**, then you call that object's **moveToInsertRow method.**

Working with results sets –DELETE ROW

- To **delete a row** you need to navigate to its position by calling the relevant method of the ResultSet object.

- To **delete the row** from both the result set and database invoke **deleteRow () on the ResultSet object.**

Accessing Metadata (1/9) - receive data

- To require information about a result set when you don't know the column names before accessing the result set, the **ResultSetMetaData interface** can provide such information:

– the number of columns in a result set

– the SQL datatypes of the columns

– the names of the columns

Accessing Metadata (2/9) – SELECT statement

- To receive all the data in the orders table, execute a SELECT statement query to return a **ResultSet object – rs:**

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   Connection con = DriverManager.getConnection(rl);
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery("SELECT * FROM ORDERS");
```

Accessing Metadata (3/9) - receive all data about the result itself

- To receive information about the result set itself, you have to create a **ResultSetMetaData object - rsm**, which incorporates all the information about the **ResultSet object - rs.**

```
ReslutSetMetData rsm = rs.getMetaData();
```

Accessing Metadata (4/9) – number of columns

- The **ResultSetMetaData** interface allow you to obtain the number of columns in a result set by invoking the **getColumnCount method** of the **ResultSetMetaData object**

- This number that can be used in a loop to iterate through the columns in a row within the result set:

```
int columnTotal = rs.getColumnCount();
rs.next();
for (int c =1; c <= columnTotal; c++) {
rs.getString(c);
                    }
```

Accessing Metadata (5/9) - getMethods for columns

- The **getColumnType method** enables you to obtain the SQL datatype of a particular column, returns the column's SQL datatype as **an integer**.

- The **getColumnTypeName method** returns the column's SQL datatype **as a string** - for example, "VARCHAR", "NUMBER".

- The **getColumnLabel method** returns a string that's the title of a particular column - for example "cust_no".

Accessing Metadata (6/9) – information about databases

•You can obtain information about the **database system** to which your application has connected using the **DatabaseMetaData interface,** which provides data source information at run time:

– connect to the **Database Management System (DBMS)** and use this connection to create a **DatabaseMetaData object,** which includes the necessary information about the data source:

```
String getDatabaseProductName()
int getMaxConnections()
boolean spportOuterJoins()
ResultSet getTables
```

Accessing Metadata (7/9) - database metadata

•In general, database metadata provides information on
– the database and JDBC driver used
– database restrictions
– supported database features

•DatabaseMetaData methods that return a string provide general database information: database's product name and version.

Accessing Metadata (8/9) – maximum number

•DatabaseMetaData methods that return an int datatype, are normally **getMaxX methods,** which enable you to find out the **maximum number of characters** allowed for:

– a statement
– cursors
– schema
– table or column names
– catalogs and procedures

Accessing Metadata (8/9) - getTables() and getProcedures()

•If you want to find out if the DBMS supports stored procedures, you need to invoke the **supportsStoredProcedures method**, wich returns a **Boolean true or false** value depending on whether the connected database supports stored procedures.

•The **DatabaseMetaData interface** provides methods, such as **getTables ()** and **getProcedures (),** that can return **ResultSet objects,** methods that can be used to list all the database's objects.

Thank you!

High performance. Delivered.

accenture

consulting technology | outsourcing