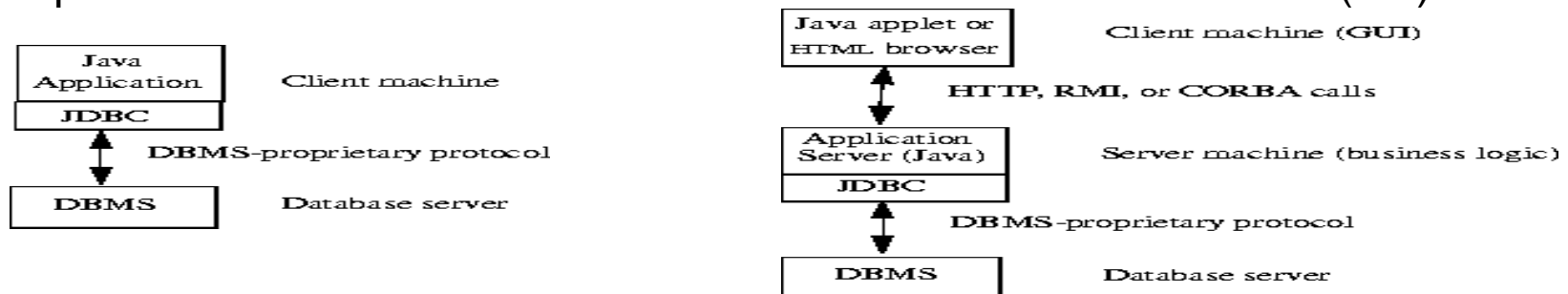


# Acceso a BD desde Java. JDBC

- **JDBC** (Java DataBase Connectivity) es una interfaz para programar la **ejecución** de **sentencias SQL** sobre **SGBDR**
- El **API JDBC** facilita programar el **acceso a BD** sin que se tenga en cuenta **a que Servidor nos dirigimos** (Oracle, Sybase, Informix, etc.).
- JDBC hace tres cosas:
  - » Establece la conexión con una BD
  - » Envía sentencias SQL
  - » Procesa los resultados

```
Connection con = DriverManager.getConnection ("jdbc:odbc:wombat", "login",  
"password");  
Statement stmt = con.createStatement();  
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");  
while (rs.next()) {  
    int x = getInt("a");String s = getString("b");float f = getFloat("c");  
}
```

- JDBC permite dos modelos de acceso a BD: de dos o de tres niveles (tier)



# Acceso a BD desde Java. JDBC

- Para que una aplicación JAVA que utilice el API JDBC pueda acceder a un **Servidor concreto** necesita un “driver” de JDBC específico para él
- Tipos de “drivers”:
  - ① Puente **JDBC-ODBC** y “driver” ODBC
  - ② “Driver” escrito en **JAVA** que hace **llamadas** al “driver” **nativo** del SGBDR
  - ③ “Driver” en **JAVA** a **Protocolo de Red independiente** del SGBDR. Después debe haber un software intermedio que traslade este protocolo a cada SGBDR particular
  - ④ “Driver” **nativo** escrito **completamente en JAVA**. Específico de cada SGBDR
- El **API JDBC** forma parte del paquete de clases de JAVA a partir del **JDK1.1x**, que incluye además:
  - » El **gestor** de “drivers” **JDBC**
  - » Una **utilidad** para evaluar que la **utilización del API JDBC sea conforme a estándares**
  - » Un **puente JDBC-ODBC** para que, si disponemos del “driver” ODBC para nuestro SGBDR, podamos escribir programas que accedan a él
- No obstante, se puede **incorporar** a un programa que utilice **JDK1.0**

# Acceso a BD desde Java. JDBC

- Veremos el empleo del API JDBC utilizando un “driver” tipo 4 de Oracle para acceso a sus servidores de BD
- Pasos para utilizar el “driver”:
  - ① Carga de las classes JDBC: `import java.sql.* ;`
  - ② Carga del “driver”: `Class.forName ("oracle.jdbc.driver.OracleDriver")`
  - ③ Conexión a la BD: `DriverManager.getConnection (arguments)`
  - ④ Enviar sentencias SQL y procesar resultados utilizando el API JDBC
- ③ La conexión se maneja mediante el método **getConnection** de la clase **DriverManager**.
- Algunas conexiones válidas para nuestro “driver”:

```
getConnection  
("jdbc:oracle:thin:<<usuario>>/<<clave>>@oracle0.ugr.es:1521: practbd");  
ó  
getConnection ("jdbc:oracle:thin:@oracle0.ugr.es:1521:practbd", "<<usuario>>",  
"<<clave>>");
```

  - » Donde, `thin` hace referencia a **nuestro driver**, `oracle0.ugr.es` es la **máquina** que alberga el servidor Oracle, `1521` es el **puerto** asignado al servicio “listener” y `practbd` es el nombre del **servidor de BD**
- ④ Existen tres tipos de sentencias:
  - ① Sentencias **SQL estáticas** (su código se conoce en tiempo de compilación) Se crean mediante el método **createStatement()** de la clase **Connection**

# Acceso a BD desde Java. JDBC

② **PreparedStatement**, creadas por el método **prepareStatement()**. Dan soporte a **SQL dinámico** (parte de la sentencia se **precompila** y parte se sustituye en **tiempo de ejecución**). Eficiente para sentencias de **ejecución reiterada**

③ **CallableStatement** creada por el método **prepareCall()** destinada a la ejecución de procedimientos almacenados

## » Creación de objetos **Statement**

```
Connection con = DriverManager.getConnection(url, "pepito", "");
Statement stmt = con.createStatement();
```

## » Métodos para la ejecución de sentencias **Statement**

- **executeQuery()**. Para consultas que devuelven un conjunto de tuplas
- **executeUpdate()**. Para Insert, Update y Delete y DDL
- **execute()**. Devuelven más de un conjunto de tuplas

» El **conjunto de tuplas** que se obtiene de aplicar el método **executeQuery()** se gestiona mediante objetos **ResultSet**

» Esta clase dispone de métodos **getxxxx(<nomb\_atributo>)** para recuperar los datos de cada atributo en el formato adecuado y del método **next()** para recorrer cada tupla del conjunto devuelto. JDBC 2.x añade los métodos **updatexxxx(<posicion\_columna>,valor)** para actualizar valores en la fila actual.

```
java.sql.Statement stmt = conn.createStatement();
ResultSet r = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (r.next())
{
    int i = r.getInt("a");
    String s = r.getString("b");
    float f = r.getFloat("c");
    System.out.println("ROW = " + i + " " + s + " " + f);
}
```

# Acceso a BD desde Java. JDBC

- » Para **consultas** de las que **ignoramos la lista y tipo de atributos** sobre los que proyectamos, disponemos del método **getMetadata()** de la clase **ResultSet**.
- » Éste **devuelve** un objeto **ResultSetMetaData** con el que podemos **obtener información** acerca de la **estructura** de las **tuplas devueltas**. Los principales métodos son:
  - **getColumnCount()** Devuelve el **número de columnas** del ResultSet
  - **getColumnLabel(int i)** Devuelve la cabecera de la columna i
  - **getColumnName(int i)** **Nombre** de la **columna** situada en la **posición i**.
  - **getColumnType(int i)** Devuelve un **número** que **identifica el tipo de dato**. La clase java.sql.Types contiene las constantes que definen cada tipo de dato sql.
  - **getColumnTypeName(int i)** Muestra en un objeto cadena el **nombre del tipo de dato** presente en la columna i.
  - **getColumnDisplaySize(int i)** El **tamaño** de visualización **en caracteres** de la columna i.
  - **getPrecision(int i)** Cuantos **dígitos decimales** tiene la columna
  - **getScale(int i)** Cuantos **dígitos** hay a la **derecha del punto decimal**
  - **getTableName(int i)** **En qué tabla está** la columna

## » Ejemplo

```
String query = "select * from Table1";
ResultSet rs = stmt.executeQuery(query);
ResultSetMetaData rsmd = rs.getMetaData();
int columnCount = rsmd.getColumnCount();
for (int i = 1; i <= columnCount; i++) {
    String s = rsmd.getColumnTypeName(i);
    System.out.println ("Column " + i + " is type " + s);}
}
```