Ciclo Formativo de Grado Superior de Administración de Sistemas Informáticos en red

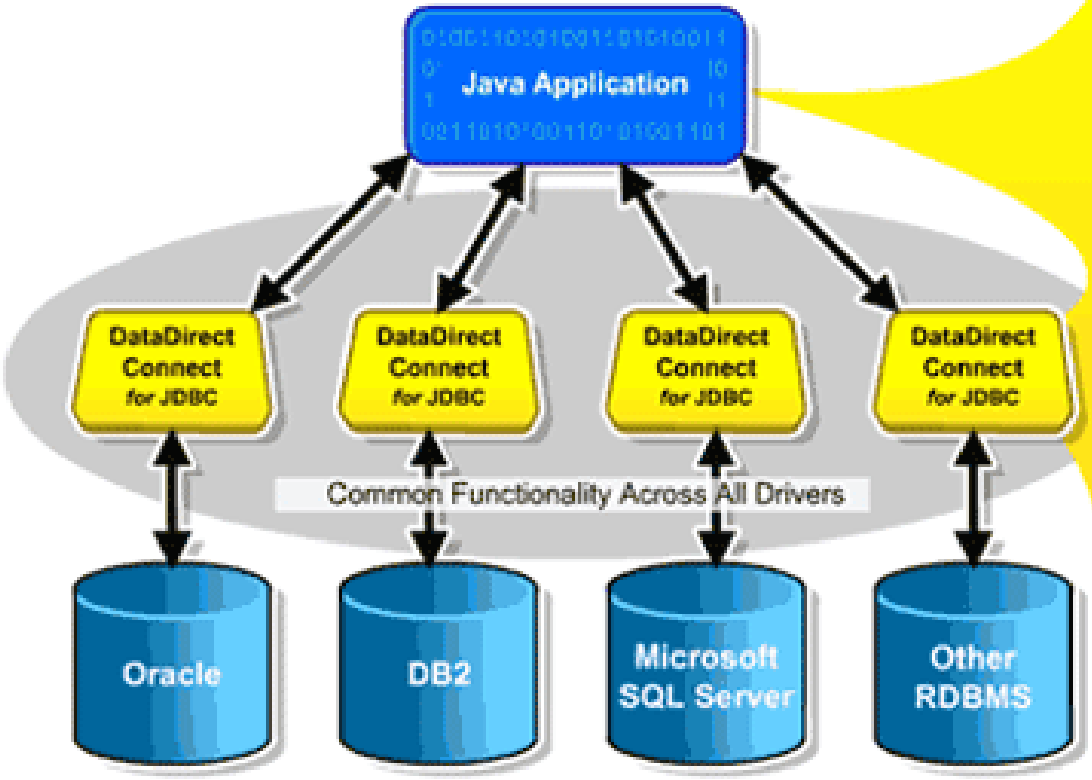
Módulo Profesional: IAW

Unidad de Trabajo 6.- Acceso a Bases de Datos desde lenguajes de <<script>> clientes y servidor.

JavaServer Pages con Java Database Connectivity (JSP/JDBC)

Departamento de Informática y Comunicación
IES San Juan Bosco (Lorca-Murcia)
Profesor: Juan Antonio López Quesada


I.E.S.
SAN JUAN BOSCO



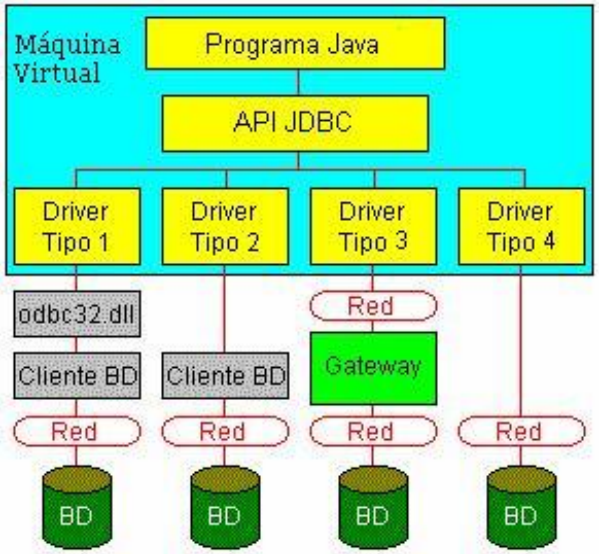
- Same application code for different databases
- Common architecture and features across all major databases, as well as database-specific features
- One driver for all Oracle versions, including 10*g*
- One driver for all DB2 backends
- Robust JDBC 3.0 implementation for improved developer productivity
- Performance and scalability leader in SPECjAppServer2002 benchmark

Índice de Contenidos


jdbc:hsql
DriverManager
setAutoCommit
connection
"UPDATE
executeUpdate
commit();
Exception e) {
rollback();

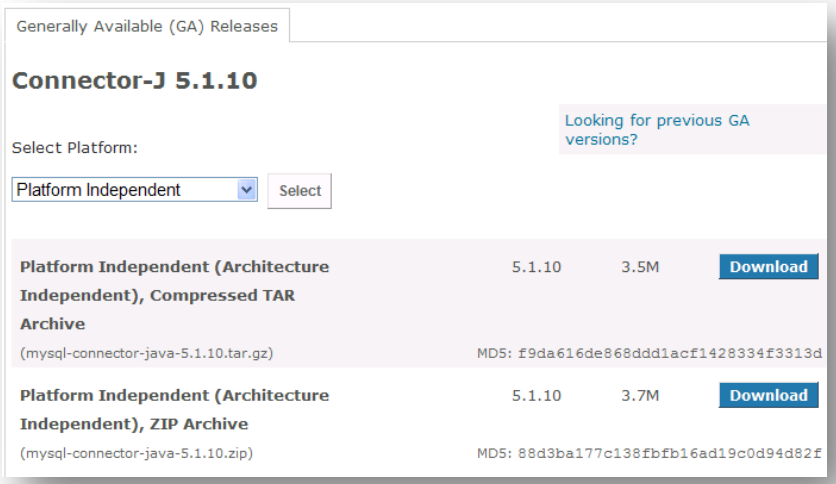
Controladores

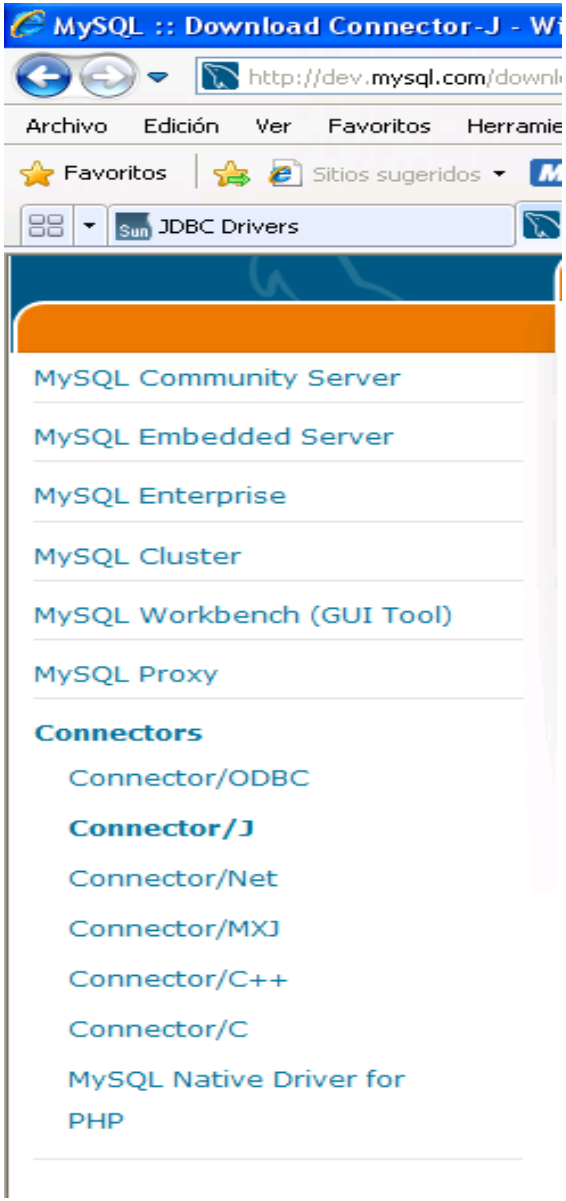
- El **API (Application Programming Interface)** que nos permite ejecutar instrucciones SQL (Structured Query Language) es distinto en cada motor de bases de datos, sin embargo los programadores de Java dispone de un solo API, la Conectividad de Bases de datos de Java (JDBC).
- Además el API JDBC 2.0 supera los límites del lenguaje SQL ya que ofrece la posibilidad de interactuar con otros tipos de orígenes de datos, como archivos que contenga tablas de datos.
- Un controlador es la implementación de la interfaz JDBC adecuada a una Base de Datos concreta.
- Con un controlador JDBC se pueden conseguir cosas:
  - ***Establecer una conexión con una fuente de datos.***
  - ***Enviar consultas e instrucciones UPDATE a la fuente de datos***
  - ***Recuperar resultados.***



Controladores

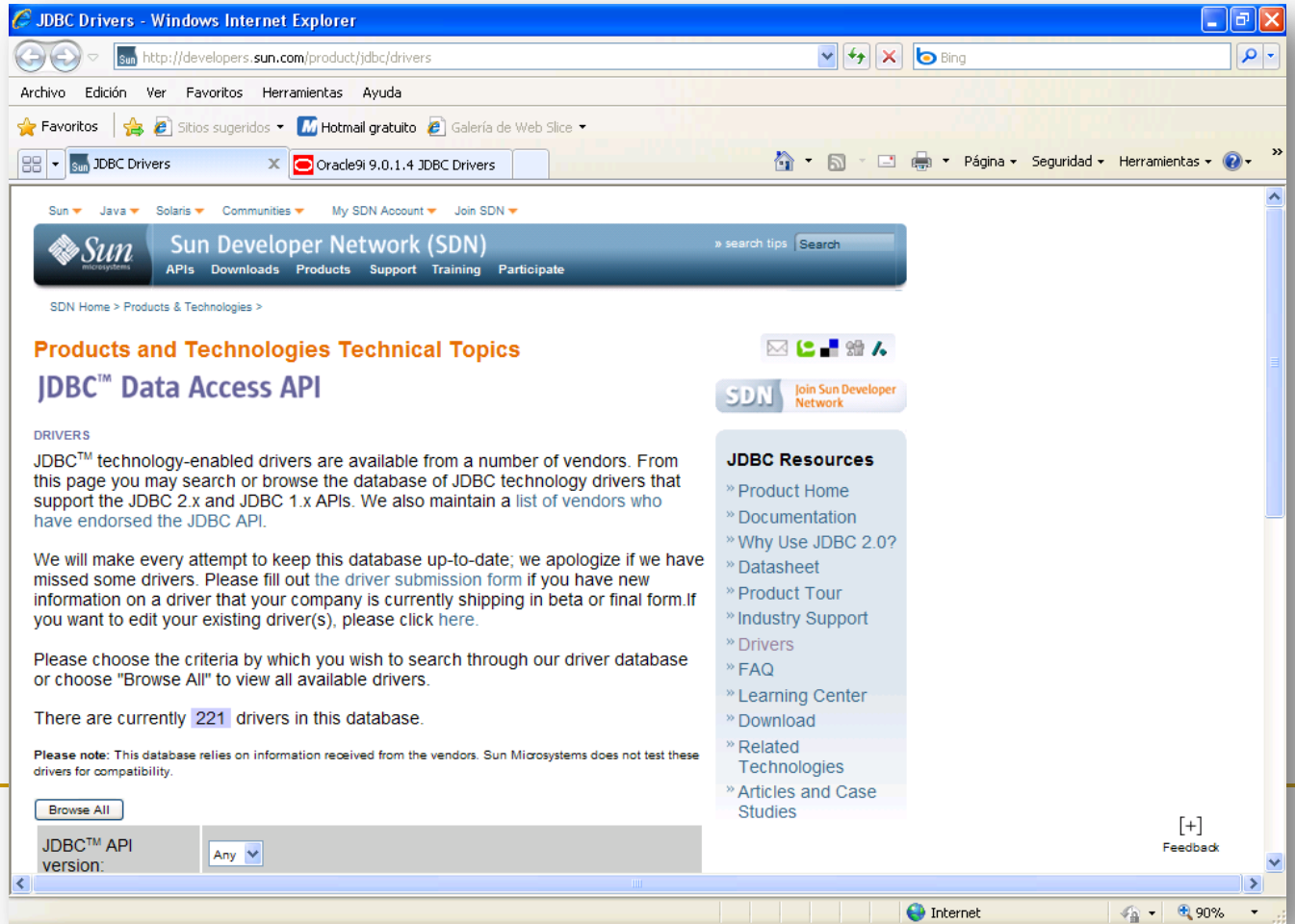
- Existen controladores desarrollados por los fabricantes para las distintasBases de Datos: Oracle, MySql, Sql*Server, Informix, … Además es posible que para una misma Base de Datos existan distintos tipos de *controladores Java.*
- *Por ejemplo, existen diversos controladores para MySQL. La información de estos controladores se puede encontrar en http://www.mysql.com/downloads.*





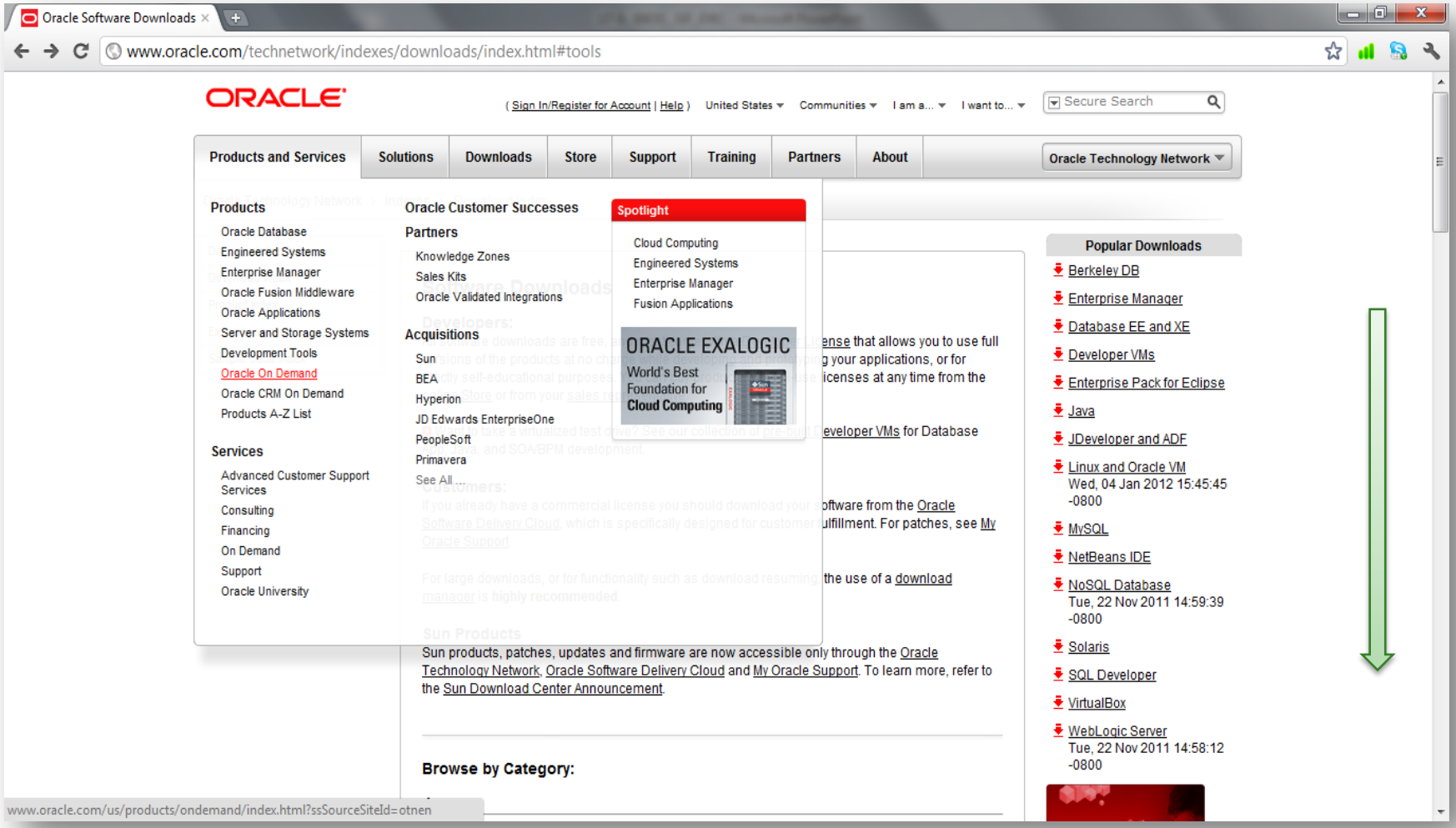
Controladores

- Es aconsejable descargar el binario del controlador y ubicarlo en el sitio correcto (como puede ser la carpeta Tomcat common/lib) y asegurarse de que la variable CLASSPATH apunta hacia dicha carpeta.
- En http://industry.java.sun.com/products/jdbc/drivers podemos encontrarla lista completa de controladores disponibles.



Controladores

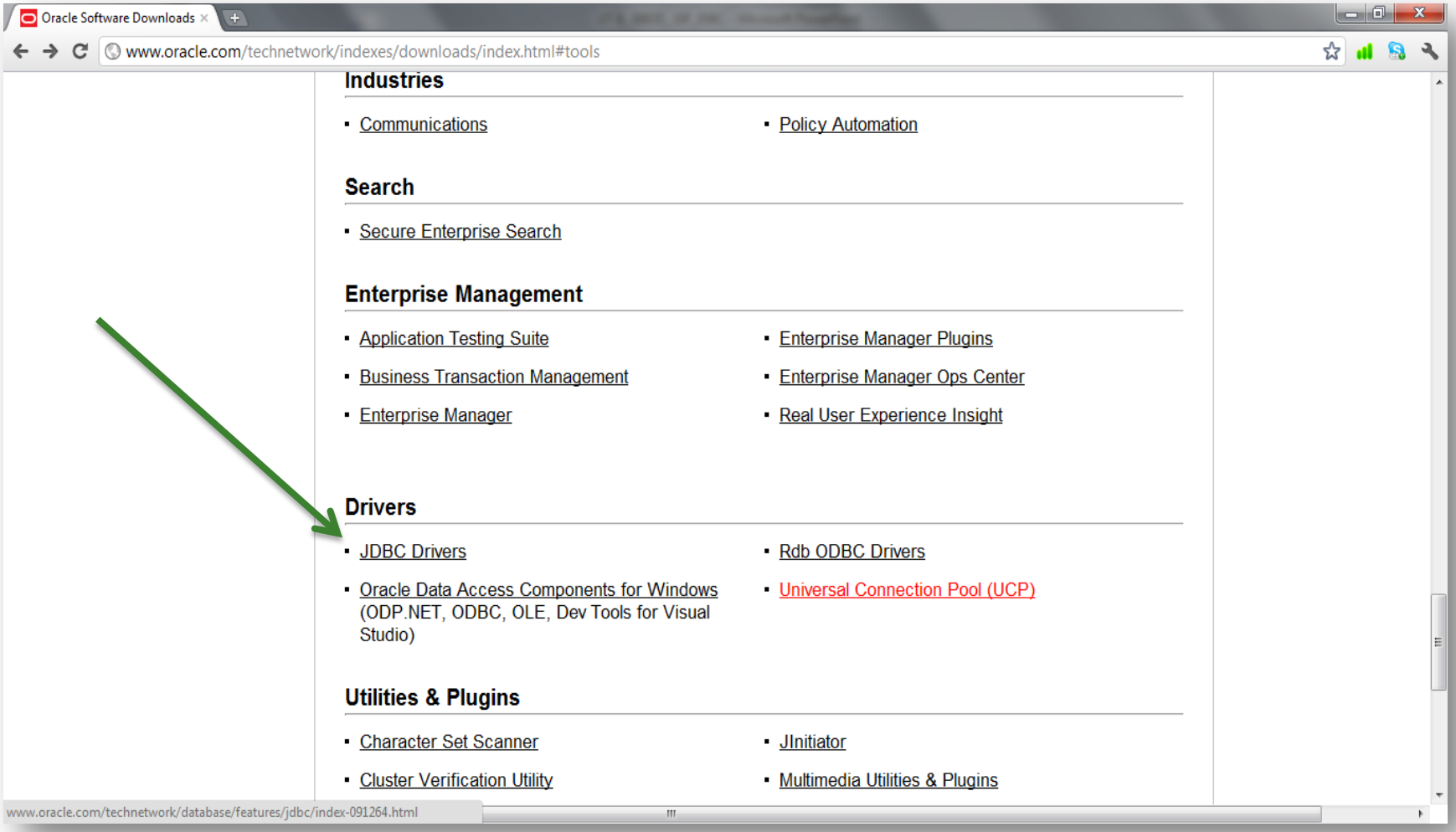
Drives JDBC Oracle.



http://www.oracle.com/technetwork/indexes/downloads/index.html#tools

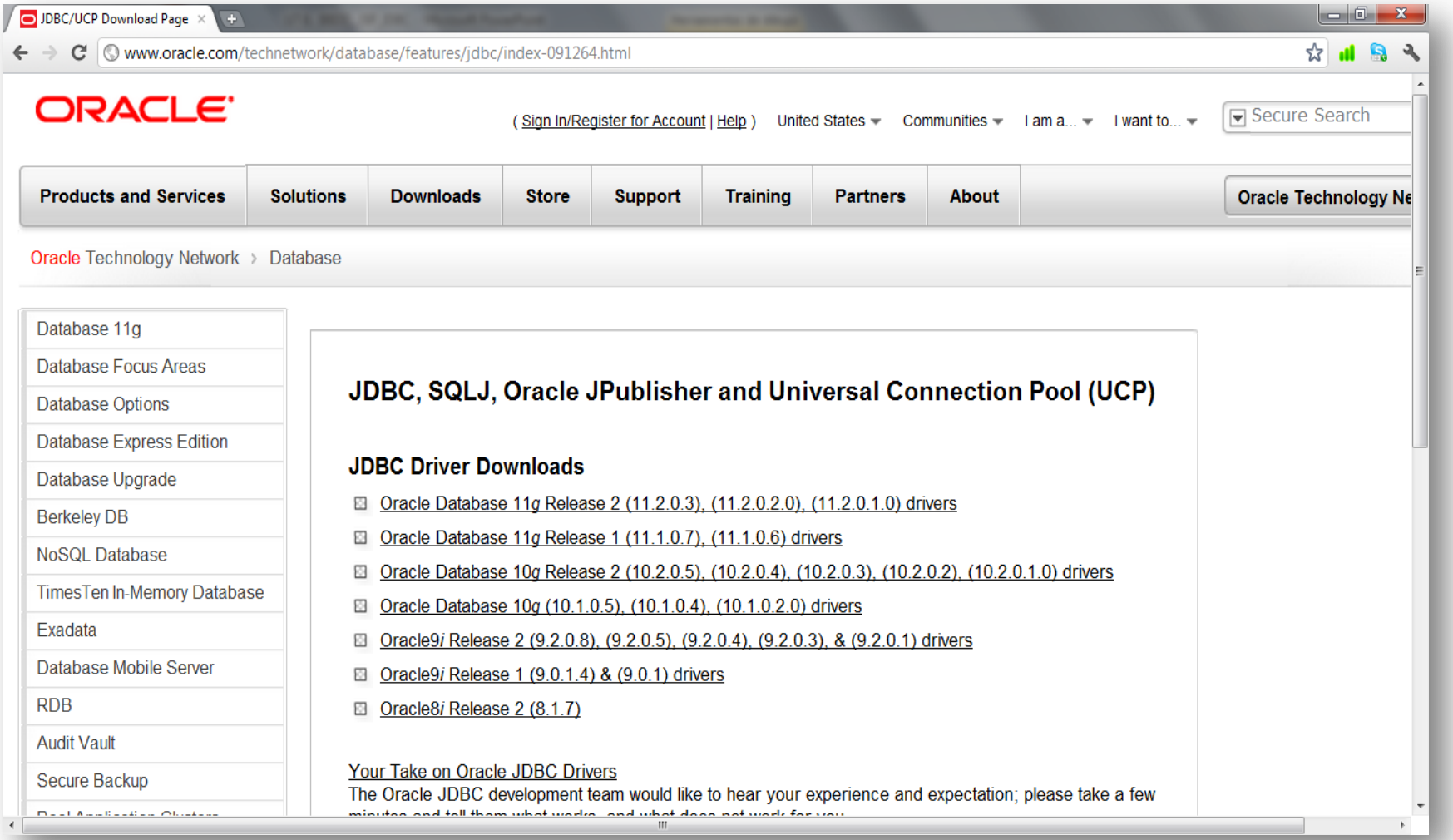
Controladores

Drives JDBC Oracle.



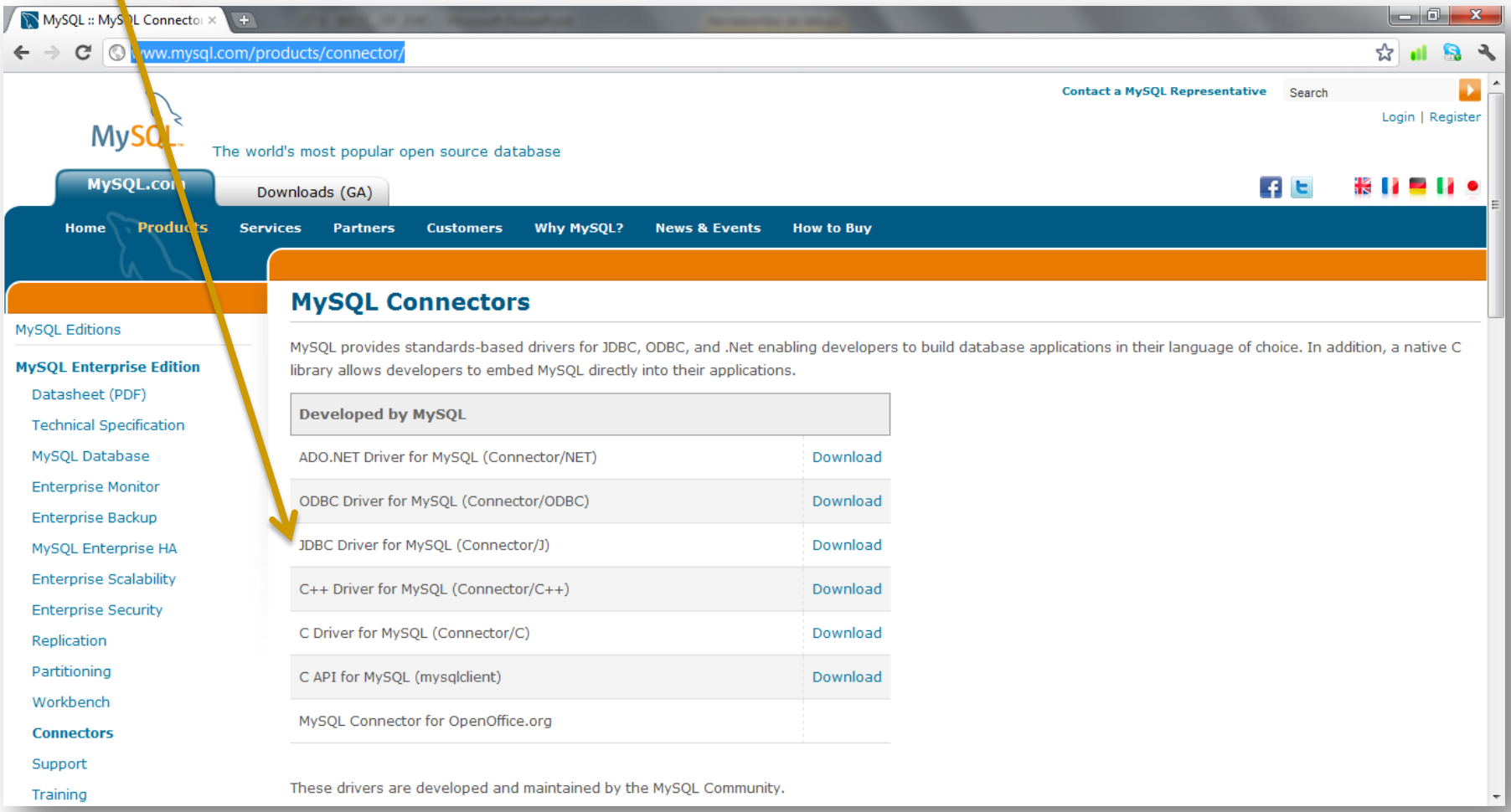
Controladores

Drives JDBC Oracle.



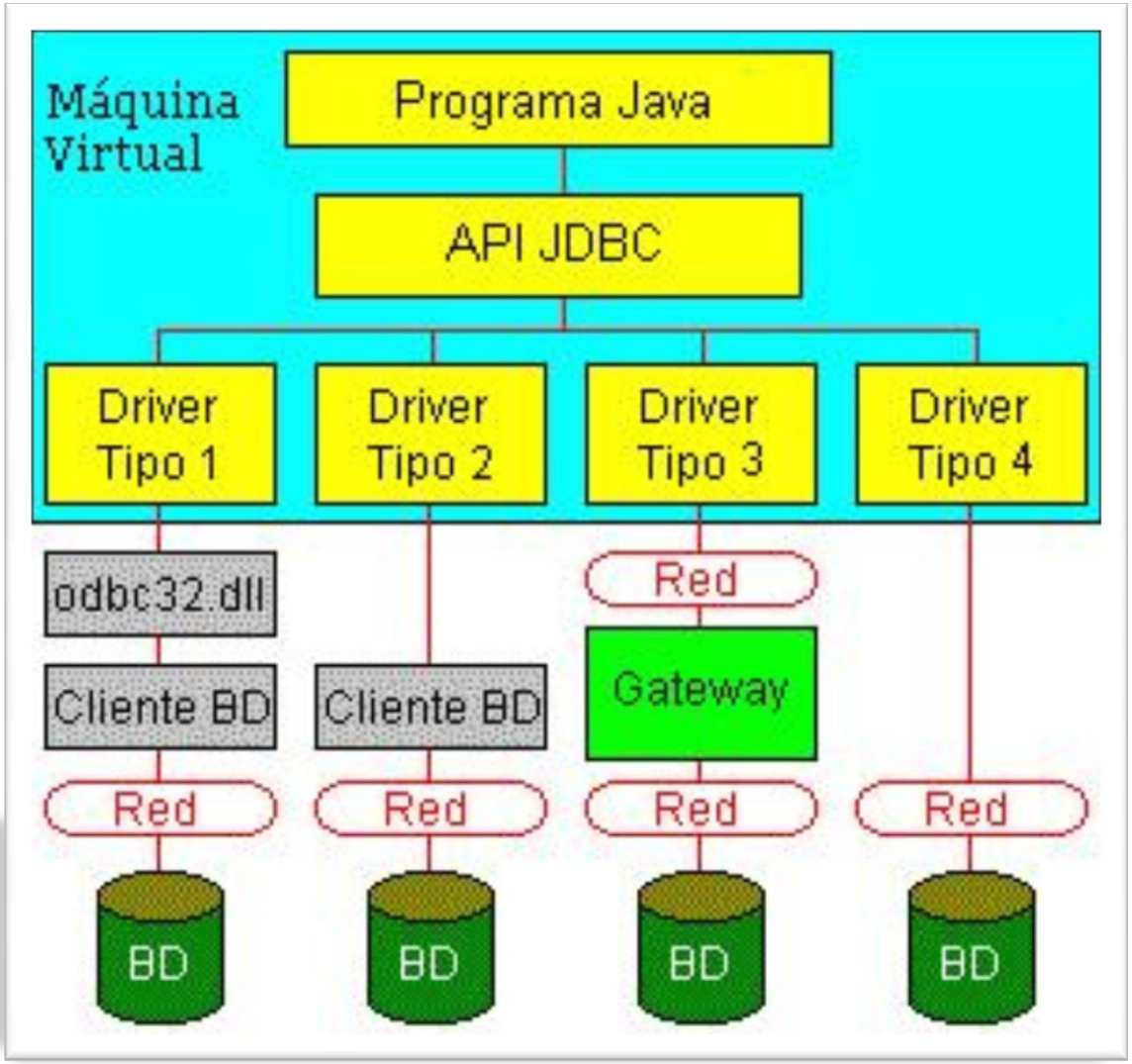
Controladores

Drives JDBC Mysql.



http://www.mysql.com/products/connector/

Tipos de controladores JDBC



Tipos de controladores JDBC

Los controladores JDBC se agrupan en cuatro categorías principales:

Tipo 1: Controlador puente JDBC-ODBC.

Ofrece acceso JDBC por medio de controladores estándar ODBC. Se usa fundamentalmente cuando la base de datos a la que se quiere acceder no cuenta con un controlador JDBC puro. No es recomendable usarlo en entornos de producción ya que el rendimiento es mucho menor.

Tipo 2: Controlador Java nativo parcialmente API.

Este tipo convierte llamadas JDBC en llamadas a métodos nativos del API de la Base de Datos como Oracle, Informix,…Al igual que en el Tipo 1 requiere código binario específico de un sistema operativo que se cargue en el equipo de cada cliente por lo que su manejo entre plataformas no es muy completo. Es más rápido que el Tipo 1.

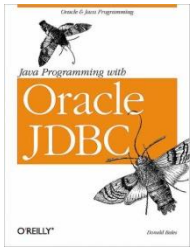
Tipo 3: Controlador Java puro de protocolo JDBC-Net.

Es la alternativa JDBC más flexible.

Tipo 4: Controlador java puro de protocolo nativo.

Convierte llamadas JDBC directamente en el protocolo de red utilizado por el DMBS. Si podemos disponer de él es aconsejable su uso.

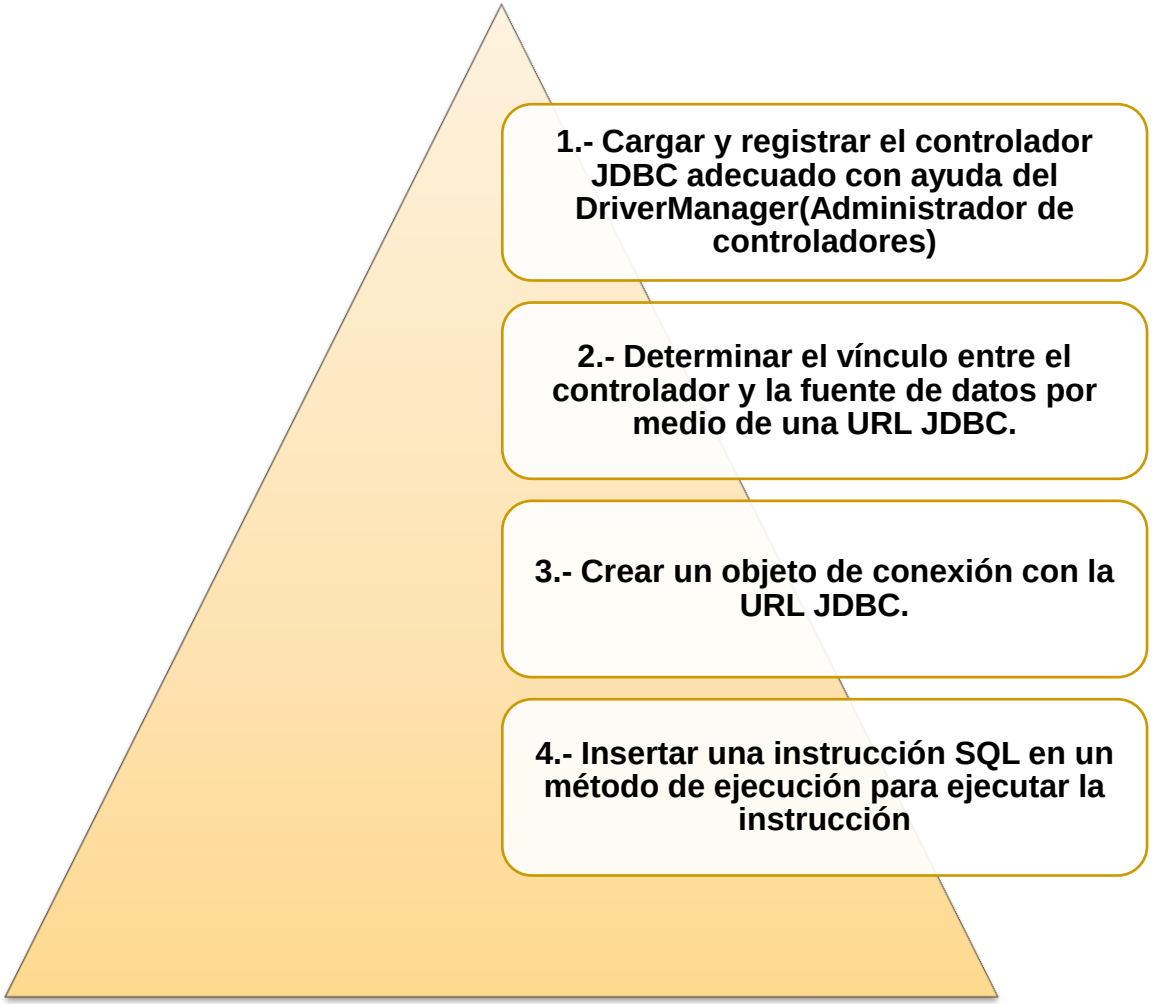
Oracle ofrece driver tipo II (Oracle OCI JDBC Driver) y driver tipo IV (Oracle Thin JDBC Driver).



Cómo acceder a una base de datos usando JDBC

TIPO	Clase JDBC
Implementación	java.sql.Driver java.sql.DriverManager java.sql.DriverPropertyInfo
Conexión a base de datos	java.sql.Connection
Sentencias SQL	java.sql.Statement java.sql.PreparedStatement java.sql.CallableStatement
Datos	java.sql.ResulSet
Errores	java.sql.SQLException java.sql.SQLWarning

Cómo acceder a una base de datos usando JDBC

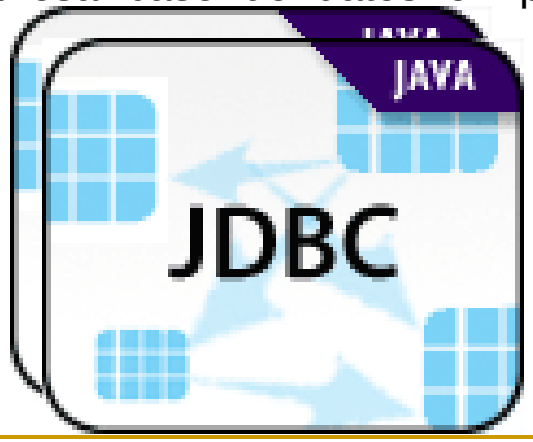


Cómo acceder a una base de datos usando JDBC - Como cargar el controlador JDBC

- Para trabajar con el API JDBC se tiene que importar el paquete **java.sql**, tal y como se indica a continuación:

  import java.sql.*;

- En este paquete se definen los objetos que proporcionan toda la funcionalidad que se requiere para el acceso a bases de datos.
- El siguiente paso después de importar el paquete java.sql consiste en cargar el controlador JDBC, es decir un objeto **Driver** específico para una base de datos que define cómo se ejecutan las instrucciones para esa base de datos en particular.



Cómo acceder a una base de datos usando JDBC - Como cargar el controlador JDBC

Hay varias formas de hacerlo, pero la más sencilla es utilizar el método **forName()** de la clase **Class**:

Class.forName("Controlador JDBC"); para el caso particular del controlador para MySQL, Connector/J, se tiene lo siguiente:

Class.forName("com.mysql.jdbc.Driver");

- *Debe tenerse en cuenta que el método estático forName() definido por la clase Class genera un objeto de la clase especificada.*
- *Cualquier controlador JDBC tiene que incluir una parte de iniciación estática que se ejecuta cuando se carga la clase.*
- *En cuanto el cargador de clases carga dicha clase, se ejecuta la iniciación estática, que pasa a registrarse como un controlador JDBC en el **DriverManager**.*

Cómo acceder a una base de datos usando JDBC - Como cargar el controlador JDBC

Es decir, el siguiente código:

Class.forName("Controlador JDBC"); es equivalente a:

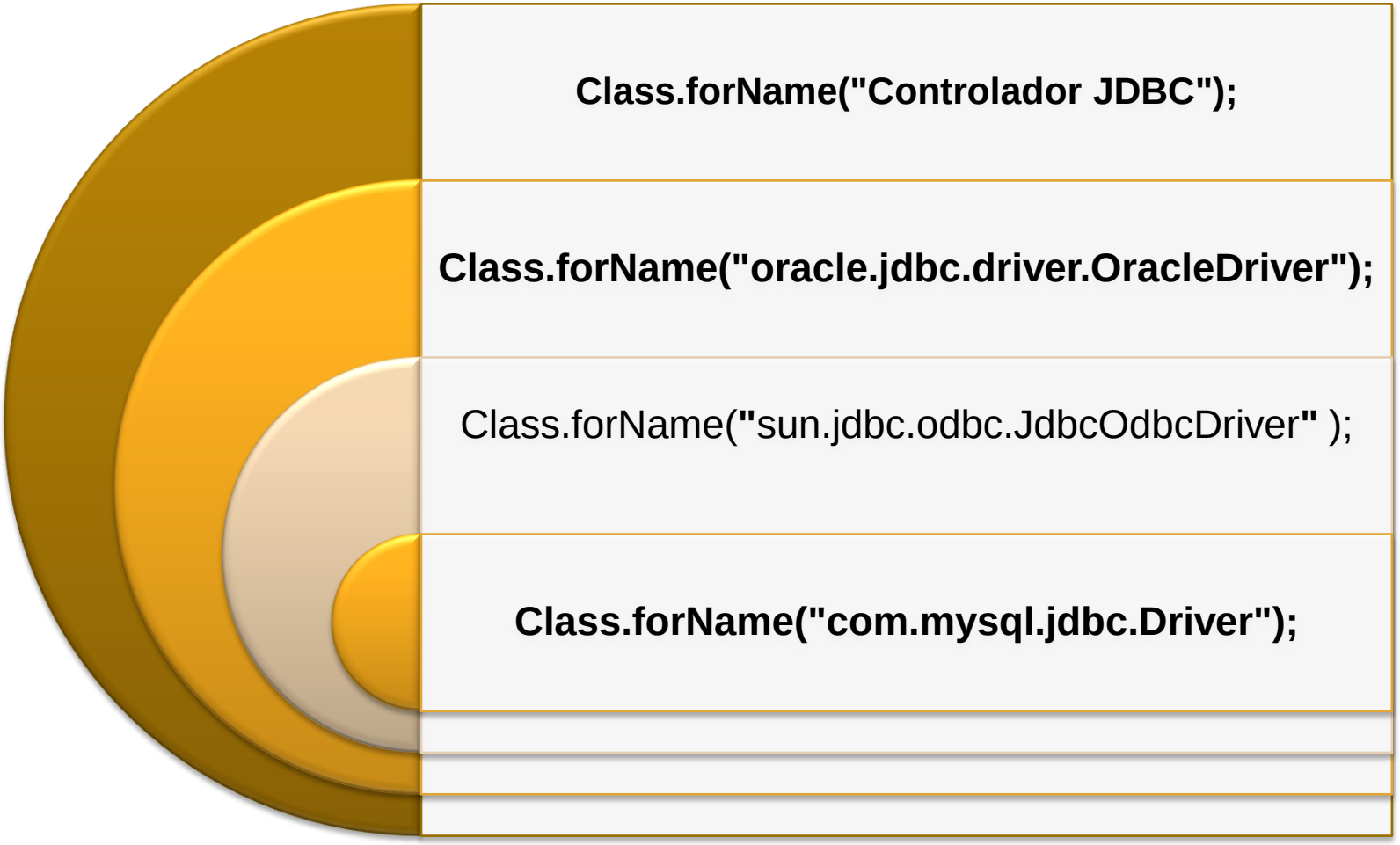
```
Class c = Class.forName("Controlador JDBC");
Driver driver = (Driver)c.newInstance();
DriverManager.registerDriver(driver);
```

Algunos controladores no crean automáticamente una instancia cuando se carga la clase. Si forName() no crea por sí solo una instancia del controlador, se tiene que hacer esto de manera explícita:

Class.forName("Controlador JDBC").newInstance(); De nuevo, para el Connector/J:

Class.forName("com.mysql.jdbc.Driver").newInstance();

Cómo acceder a una base de datos usando JDBC - Como cargar el controlador JDBC



Cómo acceder a una base de datos usando JDBC - URL de JDBC jdbc:<subprotocolo>:<data source identifier>

- Una vez registrado el controlador con el DriverManager, se debe especificar la fuente de datos a la que se desea acceder. En JDBC, una fuente de datos se especifica por medio de un URL con el prefijo de protocolo **jdbc:** , la sintaxis y la estructura del protocolo es la siguiente:

jdbc:{subprotocolo}:{subnombre}

El {subprotocolo} expresa el tipo de controlador, normalmente es el nombre del sistema de base de datos, como db2, oracle o mysql.

El contenido y la sintaxis de {subnombre} dependen del {subprotocolo}, pero en general indican el nombre y la ubicación de la fuente de datos.

Por ejemplo, para conectarse al servidor oracle "80.34.8.3 ", por "1521" que es el puerto por el que escucha el listener de la BBDD y "cicloasi" es el SID de la base de datos a la que nos conectamos.

urljdbc = "jdbc:oracle:thin:@80.34.8.3:1521:cicloasi";

urljdbc = " jdbc:mysql://80.34.8.3/name_database";

Cómo acceder a una base de datos usando JDBC - URL de JDBC jdbc:<subprotocolo>:<data source identifier>

- Una vez que se ha determinado el URL, se puede establecer una conexión a una base de datos.
- El objeto **Connection** es el principal objeto utilizado para proporcionar un vínculo entre las bases de datos y una aplicación en Java. Connection proporciona métodos para manejar el procesamiento de transacciones, para crear objetos y ejecutar instrucciones SQL, y para crear objetos para la ejecución de procedimientos almacenados.
- Se puede emplear tanto el objeto Driver como el objeto DriverManager para crear un objeto Connection. Se utiliza el método **connect()** para el objeto Driver, y el método **getConnection()** para el objeto DriverManager.
- El objeto Connection proporciona una conexión estática a la base de datos. Esto significa que hasta que se llame en forma explícita a su método **close()** para cerrar la conexión o se destruya el objeto Connection, la conexión a la base de datos permanecerá activa.
- La manera más usual de establecer una conexión a una base de datos es invocando el método getConnection() de la clase DriverManager. A menudo, las bases de datos están protegidas con nombres de usuario (login) y contraseñas (password) para restringir el acceso a las mismas. El método getConnection() permite que el nombre de usuario y la contraseña se pasen también como parámetros.

Cómo acceder a una base de datos usando JDBC - URL de JDBC jdbc:<subprotocolo>:<data source identifier>

```
String urljdbc;
String loginjdbc;
String passjdbc;
Connection conexion=null;
Statement sentencia=null;
ResultSet sentencia_sql=null;
StringBuffer built_stmt=new StringBuffer();
try
{
   Class.forName("oracle.jdbc.driver.OracleDriver");
                  //urljdbc = "jdbc:oracle:thin:@192.168.1.252:1521:cicloasi";
                  //urljdbc = "jdbc:mysql://localhost/bdtomcat ";
   urljdbc = getServletContext().getInitParameter("urljdbc");  // web.xmp Context Parametres
   loginjdbc = getServletContext().getInitParameter("loginjdbc");
   passjdbc = getServletContext().getInitParameter("passjdbc");
   conexion = DriverManager.getConnection(urljdbc,loginjdbc,passjdbc);
……
```

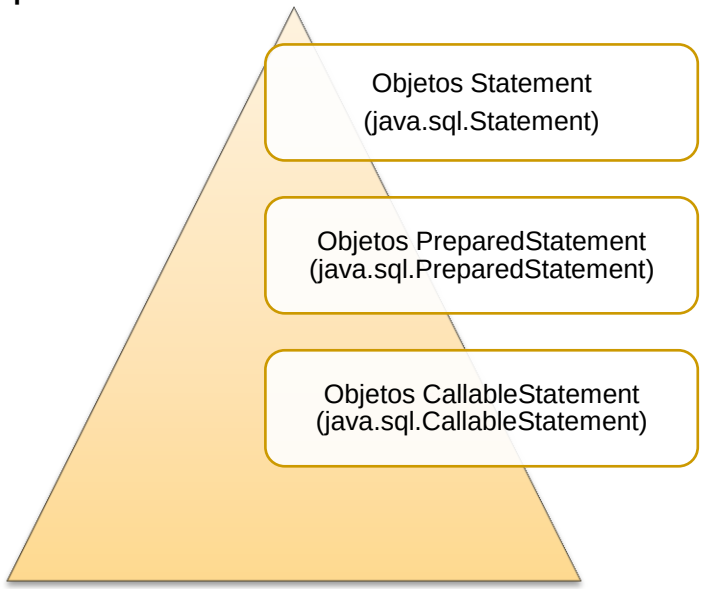
Cómo acceder a una base de datos usando JDBC - URL de JDBC jdbc:<subprotocolo>:<data source identifier>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">
    <context-param>
      <param-name>loginjdbc</param-name>
      <param-value>quesada</param-value>
   </context-param>
   <context-param>
      <param-name>passjdbc</param-name>
      <param-value>quesada</param-value>
   </context-param>
……….
```

Cómo acceder a una base de datos usando JDBC - Creación de instrucciones (objetos Statement)

- Una vez establecida la conexión a una determinada base de datos, se puede utilizar para enviar instrucciones SQL.
- Para ello, es necesario crear un objeto instrucción (Statement) que nos permita diseñar un comando SQL y ejecutarlo.
- Nos centraremos en los objetos Statement.
- Un objeto Statement se crea por medio del métod Connectionl.createStatement();veamos un ejemplo:

Statement stmt = con.createStatement();



Cómo acceder a una base de datos usando JDBC - Creación de instrucciones (objetos Statement)

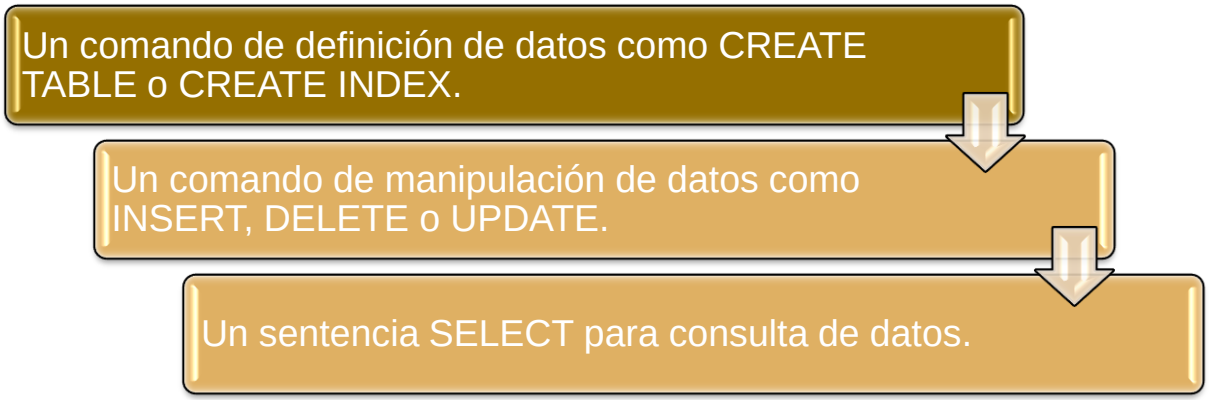
```
<%
    String urljdbc;
    String loginjdbc;
    String passjdbc;
    Connection conexion=null;
    Statement sentencia=null;
    try
    {
       Class.forName("oracle.jdbc.driver.OracleDriver");
       urljdbc = "jdbc:oracle:thin:@80.34.8.3:1521:cicloasi";
       loginjdbc = "quesada";
       passjdbc = "quesada";
       conexion = DriverManager.getConnection(urljdbc,loginjdbc,passjdbc);
       sentencia=conexion.createStatement();
       // Instrucción SQL/PLSQL
       sentencia.close();
    }
    catch (ClassNotFoundException error1)
    {
       out.println("ClassNotFoundException: No se puede localizar el Controlador de ORACLE:"
    +error1.getMessage());
    }       // Sigue en la otra diapositiva
```

Cómo acceder a una base de datos usando JDBC - Creación de instrucciones (objetos Statement)

```
catch (SQLException error2)
{
    out.println("Error en la sentencia sql que se ha intentado ejecutar (Posible error léxico y/o sintáctico):
"+error2.getMessage());
}
catch (Exception error3)
{
    out.println("Se ha producido un error indeterminado: "+error3.getMessage());
}
finally
{
    try
    {
        if (conexion != null)
            conexion.close();
    }
    catch (Exception error3)
    {
    out.println("Se ha producido una excepción finally "+ error3.getMessage());
    }
}   %>
```

Cómo acceder a una base de datos usando JDBC - Ejecución de instrucciones (Statement)

- Para ejecutar instrucciones SQL y procesar los resultados de las mismas, se debe hacer uso de un objeto Statement. Los objetos Statement envían comandos SQL a la base de datos, y pueden ser de cualquiera de los tipos siguientes:



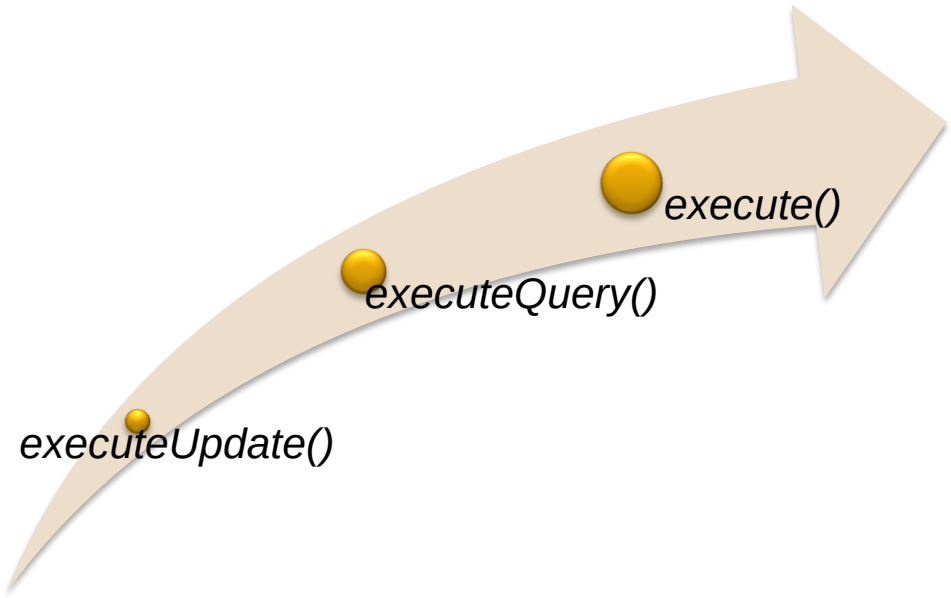
- Un comando de manipulación de datos devuelve un contador con el número de filas (registros) afectados, o modificados, mientras una instrucción SELECT devuelve un conjunto de registros denominado conjunto de resultados (resultset). La interfaz Statement no tiene un constructor , sin embargo, podemos obtener un objeto Statement al invocar el método createStatement() de un objeto Connection. Una vez creado el objeto Statement, se puede emplear para enviar consultas a la base de datos usando los métodos execute(), executeUpdate() o executeQuery().

```
conn = DriverManager.getConnection(url,login,pasword);
Statement stmt = conn.createStatement();
```

Cómo acceder a una base de datos usando JDBC - Sentencias DML , DDL .. execute(), executeUpdate()

Una vez creado el objeto Statement se puede ejecutar un comando SQL invocando distintos métodos de ejecución.

La interfaz java.sql.Statement ofrece tres métodos distintos para ejecutar instrucciones SQL:



Cómo acceder a una base de datos usando JDBC - Sentencias DML , DDL .. execute(), executeUpdate()

executeUpdate()

Se usa para ejecutar instrucciones SQL DDL como CREATE TABLE, DROP TABLE…Devuelve un entero que indica el número de filas afectadas, en el caso de CREATE TABLE y DROP TABLE devuelve 0.

execute()

Nos permite ejecutar instrucciones que devuelven más de un conjunto de resultados, más de un número de actualización o una combinación de ambos.

Valor devuelvo por el método execute():

true

Si la instrucción ejecutada devolvió un conjunto de registros (Query).

Podremos recuperar el ResultSet mediante el método getResultSet().

false

Si no fue instrucción de consulta.

Podremos recuperar la cuenta de filas afectadas con el método getUpdateCount().

Cómo acceder a una base de datos usando JDBC - Sentencias DML , DDL .. execute(), executeUpdate()

```
<%
……..
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            urljdbc = "jdbc:oracle:thin:@80.34.8.3:1521:cicloasi";
            loginjdbc = "quesada";
            passjdbc = "quesada";
            conexion = DriverManager.getConnection(urljdbc,loginjdbc,passjdbc);
            sentencia=conexion.createStatement();
            sentencia.execute("insert  ……..");
            sentencia.close();
        }
        catch (ClassNotFoundException error1)
        {
            out.println("ClassNotFoundException: No se puede localizar el Controlador de ORACLE:"
       +error1.getMessage());
        }
……..
```

Cómo acceder a una base de datos usando JDBC - Sentencias SQL-SELECT - executeQuery()

- Se usa para instrucciones que devuelven un solo conjunto de resultados, como instrucciones SELECT.
- Dicho método devuelve una instancia que implementa ResultSet. Esta estructura de datos encapsula al conjunto de registros devuelto.

RECUPERACIÓN DE LOS REGISTROS

1. Para acceder a cada uno de los registros devueltos deberemos atravesar la estructura (el cursor) con el método next(). Dicho método devuelve un booleano indicando si todavía quedan registros.

2.- Inicialmente el cursor está situado delante del primer registro, por lo que deberemos invocar primeramente next() para acceder al primer registro.

3.- Una vez situados en un registro, podremos recuperar cualquier columna con la familia de métodos getXXX() indexando la columna por:

- Posición (posición en base 1 en la cláusula SELECT).
- Nombre.

donde XXX especifica alguno de los tipos Java con el que se quiere recuperar la columna. P. ej. getString()

Cómo acceder a una base de datos usando JDBC - Sentencias SQL-SELECT - executeQuery()

```
Statement stm = conn.createStatement();
ResultSet rs = stm.executeQuery("SELECT TITLE, AUTHOR, ISBN,PRECIO FROM BOOKLIST");
while (rs.next())
{
    String titulo = rs.getString(1);
    Float n = rs.getFloat(2);
...
}
```

String titulo = rs.getString(1); obtiene el valor de la primera columna de la fila actual de rs (columna COF_NAME), convirtiéndolo a un objeto String de Java y asignándolo a "titulo".

Float n = rs.getFloat(2); : obtiene el valor de la segunda columna de la fila actual de rs, lo convierte a un float de Java y lo asigna a "n".

Recuerda que el número de columna se refiere al número de columna en la hoja de resultados (ResultSet) no en la tabla original.

Cómo acceder a una base de datos usando JDBC - Sentencias SQL-SELECT - executeQuery()

Importante: CONSIDERACIONES

- *Si la sentencia SQL ejecutada con el método executeQuery() no devolviera un conjunto de registros resultado (p. ej. INSERT INTO ...), obtendríamos una excepción SQLException.*

- *Por defecto, un objeto Statement tiene solamente un ResultSet abierto. Por tanto si queremos tener más de un ResultSet abierto simultáneamente, deberemos utilizar distintos objetos Statement's.*

Existe un objeto ResultSetMetaData que proporciona varios métodos para obtener información sobre los datos que están dentro de un objeto ResultSet. Estos métodos permiten entre otras cosas obtener de manera dinámica el número de columnas en el conjunto de resultados, así como el nombre y el tipo de cada columna.

```
ResultSet res = stmt.executeQuery("SELECT * FROM agendita");
ResultSetMetaData metadata = res.getMetaData();
```

Sentencias DML , DDL ..

```
<%
    String urljdbc;
    String loginjdbc;
    String passjdbc;
    Connection conexion=null;
    Statement sentencia=null;
    ResultSet sentencia_sql=null;
    StringBuffer built_stmt=new StringBuffer();
    try
    {
       Class.forName("oracle.jdbc.driver.OracleDriver");
       //urljdbc = "jdbc:oracle:thin:@192.168.1.252:1521:cicloasi";
       urljdbc = "jdbc:oracle:thin:@80.34.8.3:1521:cicloasi";
       loginjdbc = "quesada";
       passjdbc = "quesada";
       conexion = DriverManager.getConnection(urljdbc,loginjdbc,passjdbc);
       sentencia=conexion.createStatement();
       built_stmt.append("insert into alumnos values (102,'Frank',‘López') ");
       sentencia.execute(built_stmt.toString());
       sentencia.close();
       out.println("<br><h1>OperaciÃ³n: insert into alumnos values (100,'Pedro','lÃ³pez')!!!!!<h1>");
    }
```

Sentencias DML , DDL ..

```
        catch (ClassNotFoundException error1)
        {
            out.println("ClassNotFoundException: No se puede localizar el Controlador de ORACLE:" +error1.getMessage());
        }
        catch (SQLException error2)
        {
            out.println("Error en la sentencia sql que se ha intentado ejecutar (Posible error Léxico/Sintáctico):
"+error2.getMessage());
        }
        catch (Exception error3)
        {
            out.println("Se ha producido un error indeterminado: "+error3.getMessage());
        }
        finally
        {
            try
            {
                if (conexion != null)
                    conexion.close();
            }
            catch (Exception error3)
            {
            out.println("Se ha producido una excepción finally "+ error3.getMessage());
            }
    }
}
```

Sentencias SQL-SELECT

```
<%
        String urljdbc;
        String loginjdbc;
        String passjdbc;
        String codigo;
        String nombre;
        String apellido;
        Connection conexion=null;
        Statement sentencia=null;
        ResultSet sentencia_sql=null;
        StringBuffer built_stmt=new StringBuffer();
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            urljdbc = "jdbc:oracle:thin:@192.168.1.252:1521:cicloasi";
            //urljdbc = "jdbc:oracle:thin:@80.34.8.3:1521:cicloasi";
            loginjdbc = "quesada";
            passjdbc = "quesada";
            conexion = DriverManager.getConnection(urljdbc,loginjdbc,passjdbc);
            sentencia=conexion.createStatement(ResultSet.TYPE_FORWARD_ONLY,ResultSet.CONCUR_READ_ONLY);
            built_stmt.append("select * from alumnos");
            sentencia_sql= sentencia.executeQuery(built_stmt.toString());
            // Bucle de recorrido
```

Sentencias SQL-SELECT

```
        while (sentencia_sql.next())
        {
            codigo = sentencia_sql.getString(1);
            nombre = sentencia_sql.getString("nombre");
            apellido = sentencia_sql.getString(3);
            out.println(codigo+" "+nombre+" "+apellido+"  ");
            out.println("<br>");
        }
        sentencia_sql.close();
        sentencia.close();
        out.println("<br><h1>OperaciÃ³n !!!!!<h1>");
    }

…….. catch()……..
```

Sentencias SQL-SELECT

```
<%
        String urljdbc;
        String loginjdbc;
        String passjdbc;
        Connection conexion=null;
        Statement sentencia=null;
        Statement sentencia1=null;
        ResultSet sentencia_sql=null;
        StringBuffer built_stmt=new StringBuffer();
        StringBuffer built_stmt1=new StringBuffer();
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            urljdbc = "jdbc:oracle:thin:@192.168.1.252:1521:cicloasi";
            //urljdbc = "jdbc:oracle:thin:@80.34.8.3:1521:cicloasi";
            loginjdbc = "quesada";
            passjdbc = "quesada";
            conexion = DriverManager.getConnection(urljdbc,loginjdbc,passjdbc);
            sentencia1=conexion.createStatement(ResultSet.TYPE_FORWARD_ONLY,ResultSet.CONC
            UR_READ_ONLY);
            built_stmt1.append("select * from alumnos where codigo=101");
            sentencia_sql= sentencia1.executeQuery(built_stmt1.toString());
```

Sentencias SQL-SELECT

```
        if (!sentencia_sql.next())
        {
                sentencia=conexion.createStatement();
                built_stmt.append("insert into alumnos values (101,'Pedro',‘lópez') ");
                sentencia.execute(built_stmt.toString());
                sentencia.close();
        }
        else
        {
                 out.println("<br><h1>El usuario ya existe<h1>");
        }
        sentencia_sql.close();
        sentencia1.close();
        out.println("<br><h1>Operación !!!!!<h1>");
    }
…….. catch()……..
```

Sentencias SQL-SELECT

	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR
getByte	X	x	x	x	x	x	x	x	x	x	x
getShort	x	X	x	x	x	x	x	x	x	x	x
getInt	x	x	X	x	x	x	x	x	x	x	x
getLong	x	x	x	X	x	x	x	x	x	x	x
getFloat	x	x	x	x	X	x	x	x	x	x	x
getDouble	x	x	x	x	x	X	X	x	x	x	x
getBigDecimal	x	x	x	x	x	x	x	X	X	x	x
getBoolean	x	x	x	x	x	x	x	x	x	X	x
getString	x	x	x	x	x	x	x	x	x	x	X
getBytes											
getDate											x
getTime											x
getTimestamp											x
getAsciiStream											x
getUnicodeStream											x
getBinaryStream											
getObject	x	x	x	x	x	x	x	x	x	x	x

	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP	VARCHAR
getByte	x	x						
getShort	x	x						
getInt	x	x						
getLong	x	x						
getFloat	x	x						
getDouble	x	x						
getBigDecimal	x	x						
getBoolean	x	x						
getString	X	X	x	x	x	x	x	x
getBytes			X	X	x			
getDate	x	x				X		x
getTime	x	x					X	x
getTimestamp	x	x				x	x	X
getAsciiStream	x	X	x	x	x			
getUnicode	x	X	x	x	x			

Sentencias SQL-SELECT

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet srs = stmt.executeQuery("SELECT COF_NAME, PRICE FROM COFFEES");
```

- El primer argumento es una de las tres constantes añadidas al API ResultSet para indicar el tipo de un objeto ResultSet: TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, y TYPE_SCROLL_SENSITIVE.
- El segundo argumento es una de las dos constantes de ResultSet para especificar si la hoja de resultados es de sólo lectura o actualizable: CONCUR_READ_ONLY y CONCUR_UPDATABLE. Lo que debemos recordar aquí es que si especificamos un tipo, también debemos especificar si es de sólo lectura o actualizable.

TYPE_FORWARD_ONLY se crea una hoja de resultados no desplazable, es decir, una hoja en la que el cursor sólo se mueve hacia delante (es el valor por defecto).

TYPE_SCROLL_INSENSITIVE permite crear un Resultset desplazable. No refleja cambios.

TYPE_SCROLL_SENSITIVE permite crear un Resultset desplazable. La diferencia con el anterior es que en este segundo caso si los datos sufren modificaciones después de tener abierto el ResultSet, dichos cambios se ven reflejados en el ResultSet

CONCUR_READ_ONLY crea un ResultSet de sólo lectura.

CONCUR_UPDATABLE crea unResultSet que permite actualización.

Liberación de recursos

Al cerrar un recurso, de forma implícita se cierran todos los recursos dependientes. Así por ejemplo si cerramos una conexión, también se cerrarán las sentencias y cursores ResultSets) dependientes tal y como se muestra en el ejemplo:

```
Connection conn = DriverManager.getConnection(...);
Statement stm = conn.createStatement();
ResultSet rs = stm.executeQuery(SQL);
...
conn.close();
```

Liberación de recursos

En última instancia, el recurso se liberará cuando el recolector de basura (Garbage Collector) libere el objeto que posee el recurso. No obstante es una buena recomendación y una buena técnica de programación cerrar todos los recursos de forma explícita conforme no se necesiten:

```
Connection conn = DriverManager.getConnection(...);
Statement stm = conn.createStatement();
ResultSet rs = stm.executeQuery(SQL);
...
rs.close();
stm.close();
conn.close();
```

NOTA: No hay ningún efecto si se intenta cerrar un recurso ya cerrado.

Detalles de ResultSet

Para recuperar el valor de una columna se puede hacer tanto por su número de columna como por su nombre:

String s = rs.getString(2);

String s = rs.getString(“TITULO”);

- Si conocemos el nombre de una columna pero no su número, podemos obtener éste mediante el método findColumn().
- También es posible obtener información sobre las columnas de un ResultSet (número de columnas, tipos, y propiedades), para ello usaremos el método ResultSet.getMetaData(), que devuelve un objeto ResultSetMetaData con la información antes indicada.
- De forma predeterminada el movimiento de un ResultSet es hacia delante y es el único movimiento posible con los controladores que implementan el API JDBC 1.0
- Este tipo de ResultSet devuelve el valor ResultSet.TYPE_FORWARD_ONLY del método getType() y se conoce como conjunto de resultados sólo hacia delante.

Detalles de ResultSet

Los controladores que implementan API 2.0 permiten conjuntos de resultados desplazables. Es decir es posible desplazar hacia delante y hacia atrás así como hasta una determinada fila. Los métodos para realizar estas acciones son: ***previous(), first(), last(), absolute(), relative(), afterLast() y beforeFirst().***

El método absolute() moverá el cursor al número de fila indicado en su argumento.

Si el número es positivo, el cursor se mueve al número dado desde el principio, por eso llamar a absolute(1) pone el cursor en la primera fila.

Si el número es negativo, mueve el cursor al número dado desde el final, por eso llamar a absolute(-1) pone el cursor en la última fila.

La siguiente línea de código mueve el cursor a la cuarta fila de srs.

```
srs.absolute(4);
```

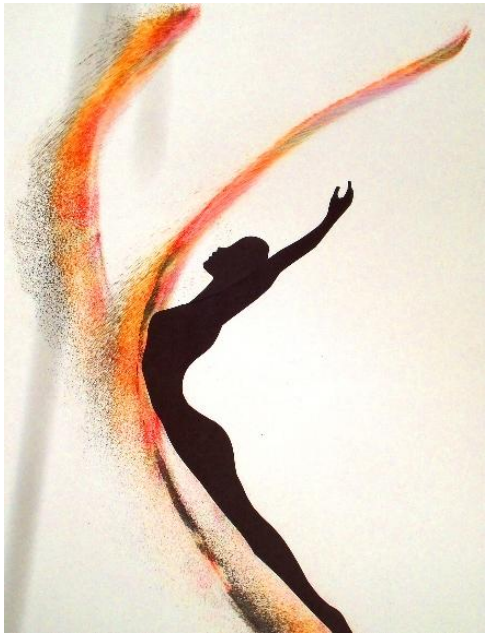
Si srs tuviera 500 filas, la siguiente línea de código movería el cursor a la fila 497.

```
srs.absolute(-4);
```

Detalles de ResultSet

Con el método relative(), se puede especificar cuántas filas se moverá desde la fila actual y también la dirección en la que se moverá. Un número positivo mueve el cursor hacia adelante el número de filas dado; un número negativo mueve el cursor hacia atrás el número de filas dado. Por ejemplo, en el siguiente fragmente de código, el cursor se mueve a la cuarta fila, luego a la primera y por último a la tercera.

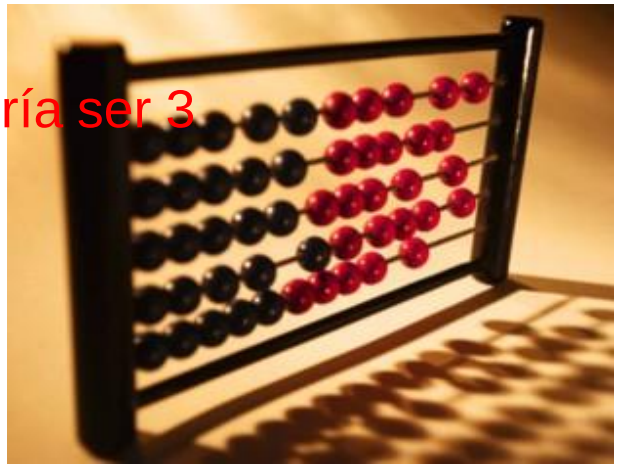
```
srs.absolute(4); // cursor está en la cuarta fila
. . .
srs.relative(-3); // cursor está en la primera fila
. . .
srs.relative(2); // cursor está en la tercera fila
```



Detalles de ResultSet

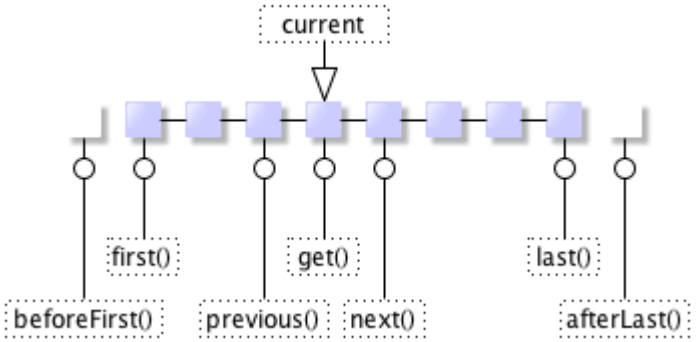
El método getRow permite comprobar el número de fila donde está el cursor. Por ejemplo, se puede utilizar getRow para verificar la posición actual del cursor en el ejemplo anterior.

```
srs.absolute(4);
int rowNum = srs.getRow(); // rowNum debería ser 4
srs.relative(-3);
int rowNum = srs.getRow(); // rowNum debería ser 1
srs.relative(2);
int rowNum = srs.getRow(); // rowNum debería ser 3
```



Detalles de ResultSet

- Existen cuatro métodos adicionales que permiten verificar si el cursor se encuentra en una posición particular.
- La posición se indica en sus nombres: isFirst, isLast, isBeforeFirst, isAfterLast. Todos estos métodos devuelven un boolean y por lo tanto pueden ser utilizados en una sentencia condicional.
- Por ejemplo, el siguiente fragmento de código comprueba si el cursor está después de la última fila antes de llamar al método previous en un bucle while. Si el método isAfterLast devuelve false, el cursor no estará después de la última fila, por eso se llama al método afterLast. Esto garantiza que el cursor estára después de la última fila antes de utilizar el método previous en el bucle while para cubrir todas las filas de srs.



Detalles de ResultSet

```
if (srs.isAfterLast() == false)
{
        srs.afterLast();
}
while (srs.previous())
{
    String name = srs.getString("COF_NAME");
    float price = srs.getFloat("PRICE");
    System.out.println(name + " " + price);
}
```

Antes de poder aprovechar estas ventajas, necesitamos crear un objeto ResultSet Scrollable.

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet srs = stmt.executeQuery("SELECT COF_NAME, PRICE FROM COFFEES");
```

Detalles de ResultSet - ResultSets Actualizables

Una actualización es la modificación del valor de una columna de la fila actual. Supongamos que queremos aumentar el precio del café "French Roast Decaf" a 10.99. utilizando el API JDBC 1.0, la actualización podría ser algo como esto.

```
stmt.executeUpdate("UPDATE COFFEES SET PRICE = 10.99" +
"WHERE COF_NAME = FRENCH_ROAST_DECAF");
```

Detalles de ResultSet - ResultSets Actualizables

El siguiente fragmento de código muestra otra forma de realizar la actualización, esta vez utilizando el API JDBC 2.0.

```
uprs.last();
uprs.updateFloat("PRICE", 10.99);
```

Los métodos updateXXX de ResultSet toman dos parámetros: la columna a actualizar y el nuevo valor a colocar en ella. Al igual que en los métodos getXXX de ResultSet., el parámetro que designa la columna podría ser el nombre de la columna o el número de la columna. Existe un método updateXXX diferente para cada tipo (updateString, updateBigDecimal, updateInt, etc.)

Para que la actualización tenga efecto en la base de datos y no sólo en la hoja de resultados, debemos llamar al método updateRow de ResultSet. Ejemplo:

```
uprs.last();
uprs.updateFloat("PRICE", 10.99);
uprs.updateRow();
```

Detalles de ResultSet - ResultSets Actualizables

Si hubiéramos movido el cursor a una fila diferente antes de llamar al método updateRow, la actualización se habría perdido. Si, por el contrario, nos damos cuenta de que el precio debería haber sido 10.79 en vez de 10.99 podríamos haber cancelado la actualización llamando al método cancelRowUpdates. Tenemos que llamar al método cancelRowUpdates antes de llamar al método updateRow; una vez que se llama a updateRow, llamar a cancelRowUpdates no hará nada. Observa que cancelRowUpdates cancela todas las actualizaciones en una fila, por eso, si había muchas llamadas a método updateXXX en la misma fila, no podemos cancelar sólo una de ellas. El siguiente fragmento de código primero cancela el precio 10.99 y luego lo actualiza a 10.79.

```
uprs.last();
uprs.updateFloat("PRICE", 10.99);
uprs.cancelRowUpdates();
uprs.updateFloat("PRICE", 10.79);
uprs.updateRow();
```

Detalles de ResultSet - Insertar y Borrar Filas en un ResultSet

- Insertar se puede hacer sin comandos SQL utilizando los métodos de ResultSet del API JDBC 2.0. Básicamente, después de tener un objeto ResultSet con los resultados de la tabla COFFEES, podemos construir una nueva fila insertándola tanto en la hoja de resultados como en la tabla COFFEES en un sólo paso.
- Se construye una nueva fila en una llamada "fila de inserción", una fila especial asociada con cada objeto ResultSet. Esta fila realmente no forma parte de la hoja de resultados; podemos pensar en ella como un buffer separado en el que componer una nueva fila.
- El primer paso será mover el cursor a la fila de inserción, lo que podemos hacer llamando al método moveToInsertRow. El siguiente paso será seleccionar un valor para cada columna de la fila. Veamos un ejemplo:

Detalles de ResultSet - Insertar y Borrar Filas en un ResultSet

```
Connection con = DriverManager.getConnection("jdbc:mySubprotocol:mySubName");
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSet uprs = stmt.executeQuery("SELECT * FROM COFFEES");
uprs.moveToInsertRow(); //MUEVE A UNA FILA DE INSERCIÓN
uprs.updateString("COF_NAME", "Kona");
uprs.updateInt("SUP_ID", 150);
uprs.updateFloat("PRICE", 10.99);
uprs.updateInt("SALES", 0);
uprs.updateInt("TOTAL", 0);
uprs.insertRow();
```

Detalles de ResultSet - Insertar y Borrar Filas en un ResultSet

- Borrar una fila es la tercera forma de modificar un objeto ResultSet, y es la más simple. Todo lo que tenemos que hacer es mover el cursor a la fila que queremos borrar y luego llamar al método deleteRow. Por ejemplo, si queremos borrar la cuarta fila de la hoja de resultados uprs, nuestro código se parecería a esto.

```
uprs.absolute(4);
uprs.deleteRow();
```

- La cuarta fila ha sido eliminada de uprs y de la base de datos.
- El único problema con las eliminaciones es lo que ResultSet realmente hace cuando se borra una fila. Con algunos driver JDBC, una línea borrada es eliminada y ya no es visible en una hoja de resultados.
- Algunos drives JDBC utilizan una fila en blanco en su lugar pone (un "hole") donde la fila borrada fuera utilizada. Si existe una fila en blanco en lugar de la fila borrada, se puede utilizar el método absolute con la posición original de la fila para mover el cursor, porque el número de filas en la hoja de resultados no ha cambiado.

Detalles de ResultSet - Insertar y Borrar Filas en un ResultSet

- El método **refreshRow** para obtener los últimos valores de una fila en la base de datos.
- Este método puede utilizar muchos recursos, especialmente si el controlador de la base de datos devuelve múltiples filas cada vez que se llama a refreshRow. De todas formas, puede utilizarse cuando es crítico tener los últimos datos.
- Incluso aunque una hoja de resultados sea sensible y los cambios sean visibles, una aplicación no podría ver siempre los últimos cambios si el driver recupera varias filas a la vez y las almacena.
- Por eso, utilizar el método refreshRow es el único método para asegurarnos que estamos viendo los últimos datos.

Detalles de ResultSet - Insertar y Borrar Filas en un ResultSet

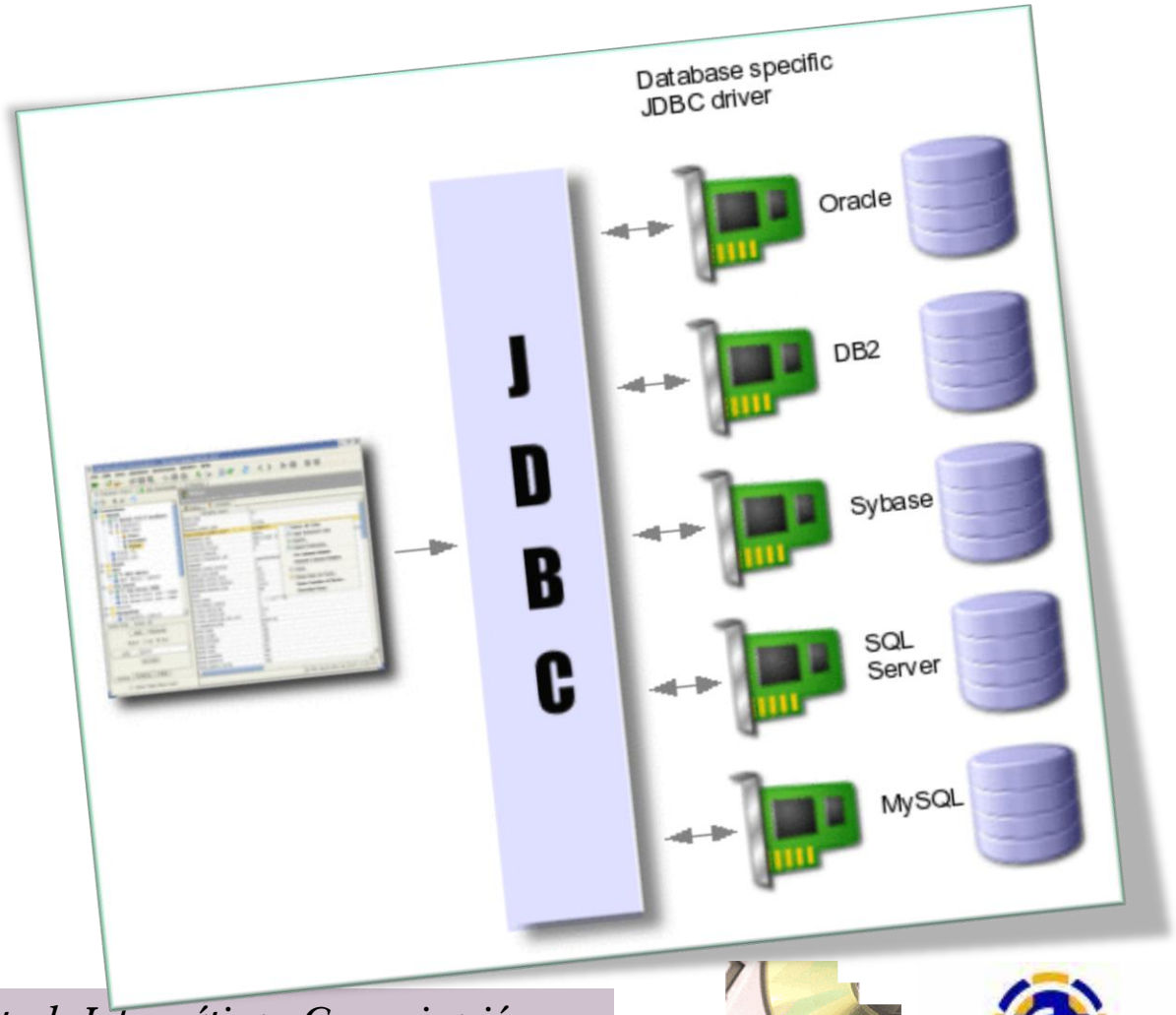
Ejemplo:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSet uprs = stmt.executeQuery(SELECT COF_NAME, PRICE FROM COFFEES);
uprs.absolute(4);
Float price1 = uprs.getFloat("PRICE");
// do something. . .
uprs.absolute(4);
uprs.refreshRow();
Float price2 = uprs.getFloat("PRICE");
if (price2 > price1)
 {
// do something. . .
}
```

Utilizar Transacciones

- Hay veces que no queremos que una sentencia tenga efecto a menos que otra también suceda.
- Una transación es un conjunto de una o más sentencias que se ejecutan como una unidad, por eso o se ejecutan todas o no se ejecuta ninguna.
- Cuando se crea una conexión, está en modo auto-entrega. Esto significa que cada entencia SQL individual es tratada como una transación.
- La forma de permitir que dos o más sentencia sean agrupadas en una transacción es desactivar el modo auto-entrega. Esto se demuestra en el siguiente código, donde con es una conexión activa.

```
con.setAutoCommit(false);
…..
con.commit()/rollback();
con.setAutoCommit(true);
```



Departamento de Informática y Comunicación
IES San Juan Bosco (Lorca-Murcia)
Profesor: Juan Antonio López Quesada

FORMACION PROFESIONAL