UTN SF DISTANCIA

# OTRA FORMA DE APRENDER

**AREA:**
Diseño y Desarrollo de Sistemas

**CURSO:**
Introducción al Lenguaje Java

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL SANTA FE
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA
TE +54 ( 0342) 4696066 / (0342) 4696432
http://otraformadeaprender.frsf.utn.edu.ar
infocursos@frsf.utn.edu.ar

# API JDBC : Acceso a Bases de datos

# Objetivos

- Introducción a JDBC
- Arquitectura de Acceso a Datos
- Tipos de drivers
  - Puente JDBC a ODBC
  - Java/Binario o Java/Código Nativo
  - Java/Protocolo nativo
  - JDBC/Net
- API JDBC
- Requisitos para el uso de JDBC
- Como acceder a la base de datos
- Recuperar Resultados
- Transacciones

# Java Database Conectivity (JDBC)

- Interfaz estándar para acceder y procesar datos almacenados en BD relacionales desde Java.

- JDBC es usado para enviar comandos SQL hacia una base de datos independientemente del DBMS (DataBase Management System)
  - Oracle
  - Infomix
  - Sybase
  - MySql
  - MSQLServer
  - etc.

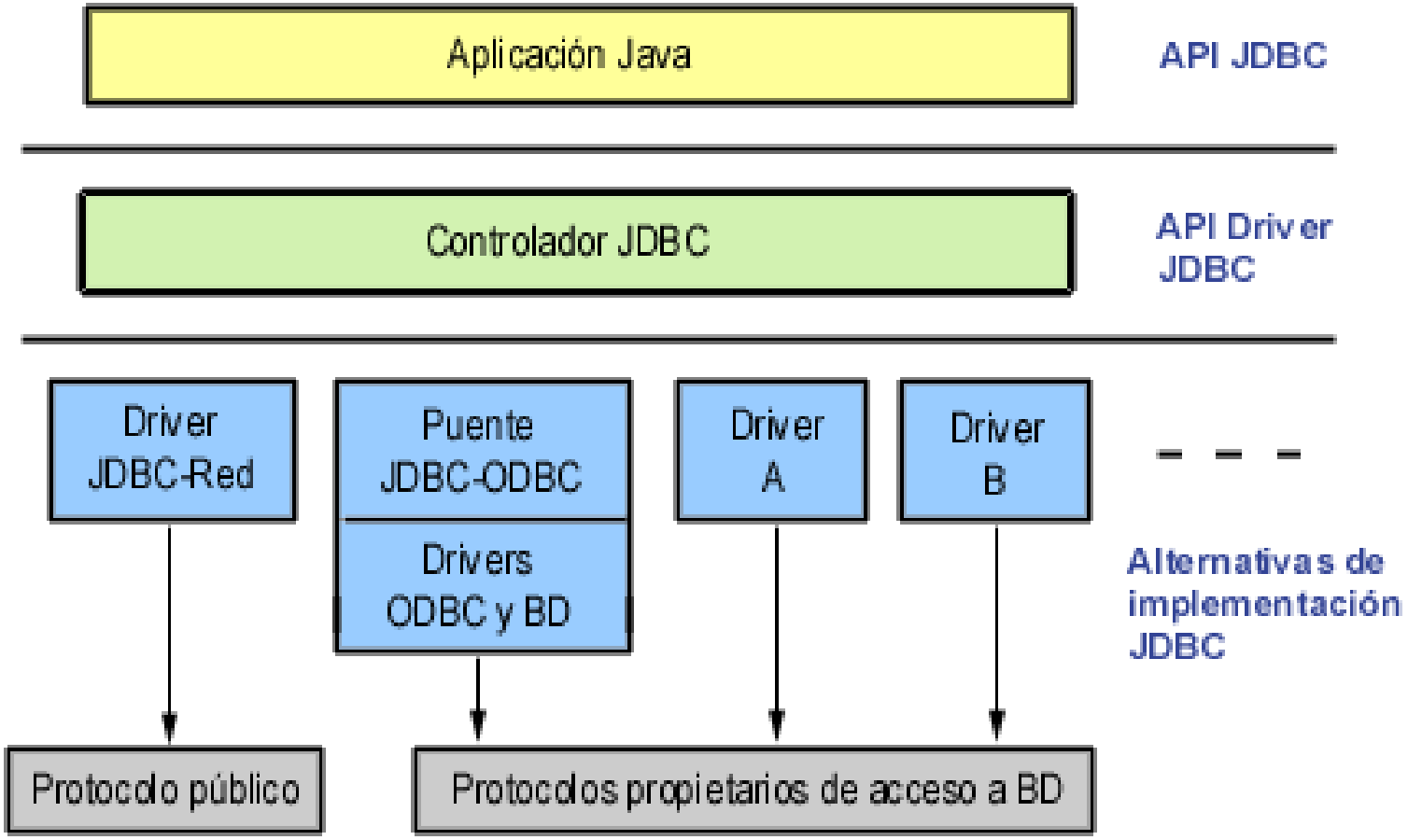
# Java Database Conectivity (JDBC)

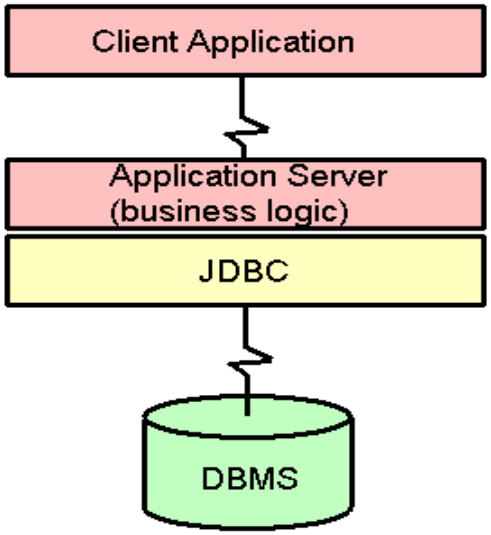
- La librería de clases JDBC es parte de Java Standard Edition (J2SE)

- JDBC es utilizado por aplicaciones cliente, applets, servlets y en contenedores dentro de servidores.

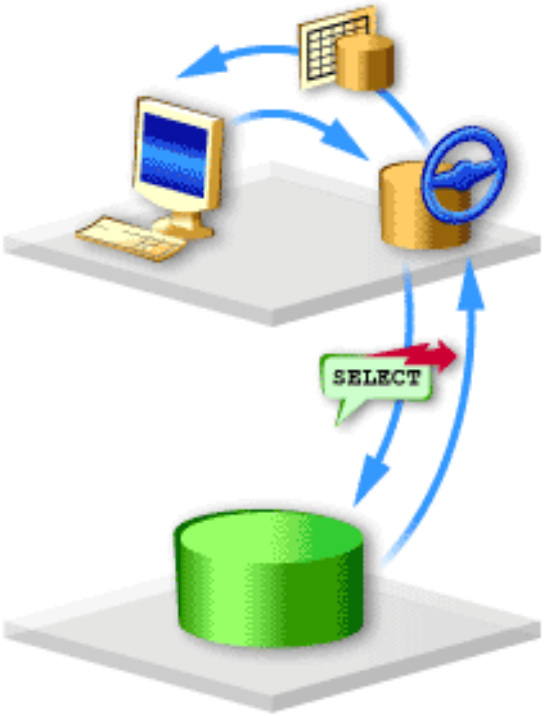
- JDBC es:
  - Una especificación de un conjunto de clases y métodos...
  - ... que permiten a cualquier programa Java acceder a bases de datos de forma homogénea.

# Conectividad JDBC



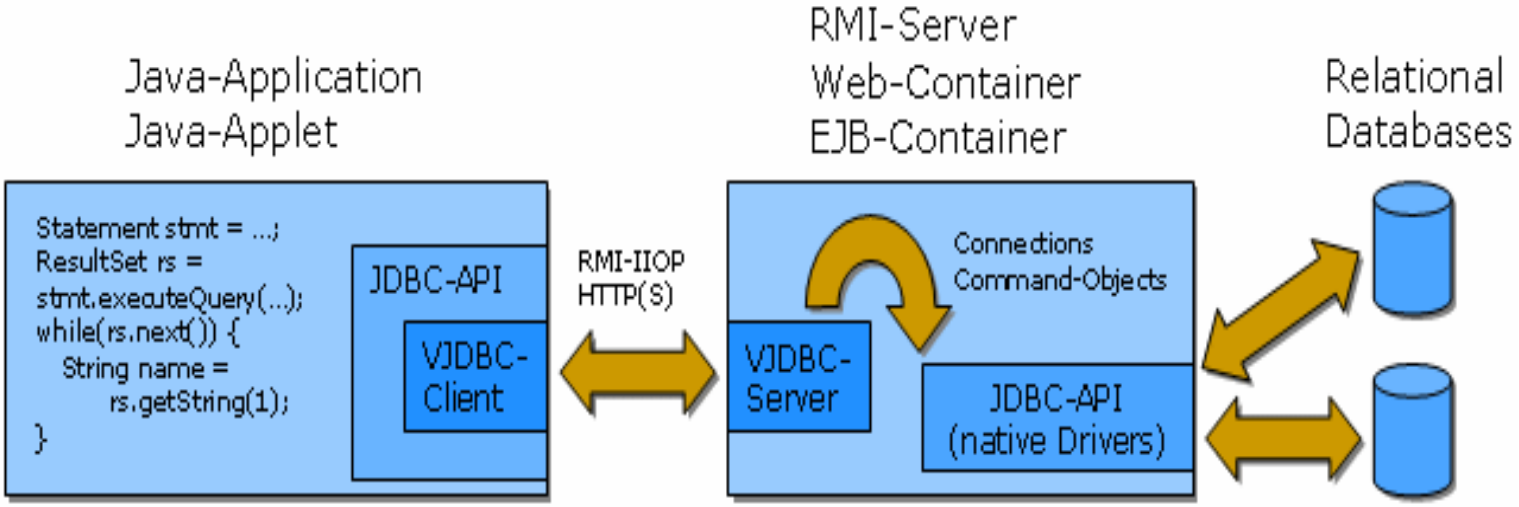



**Client Tier**
The client is the first tier in the two-tier model.



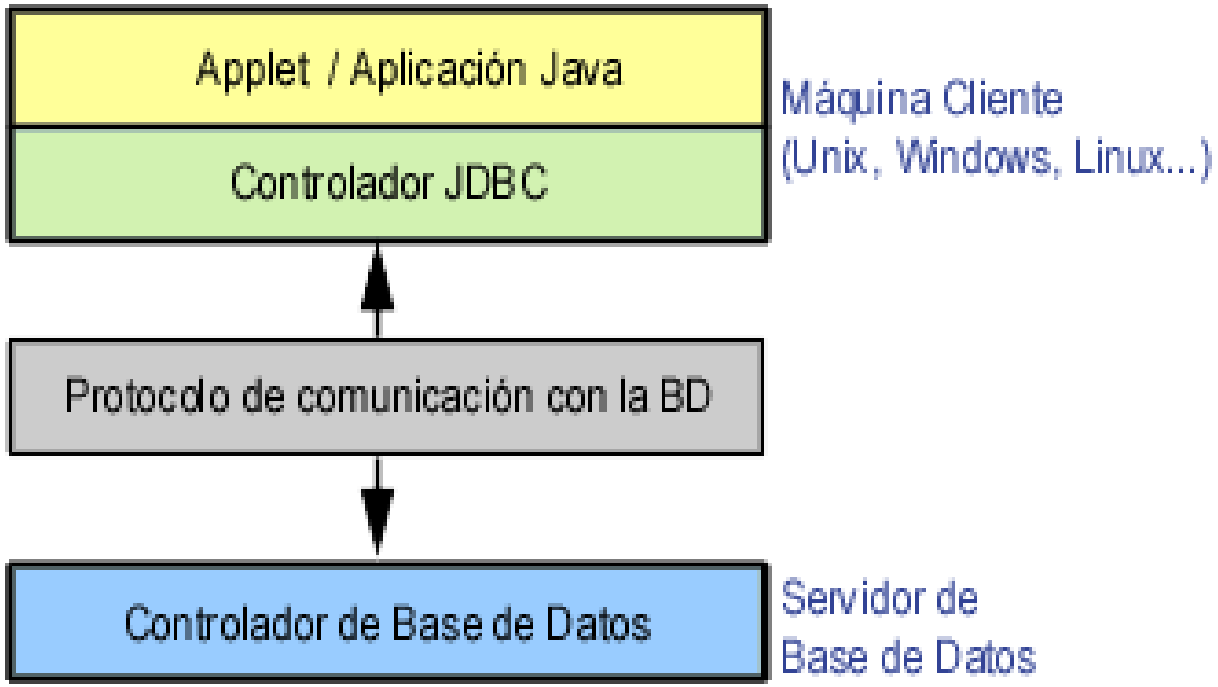

**JDBC Driver**
Provides the request/response communication between the two tiers.

**DBMS Tier**
This is the persistent datastore (e.g., a database) which may reside on the same computer as the client.



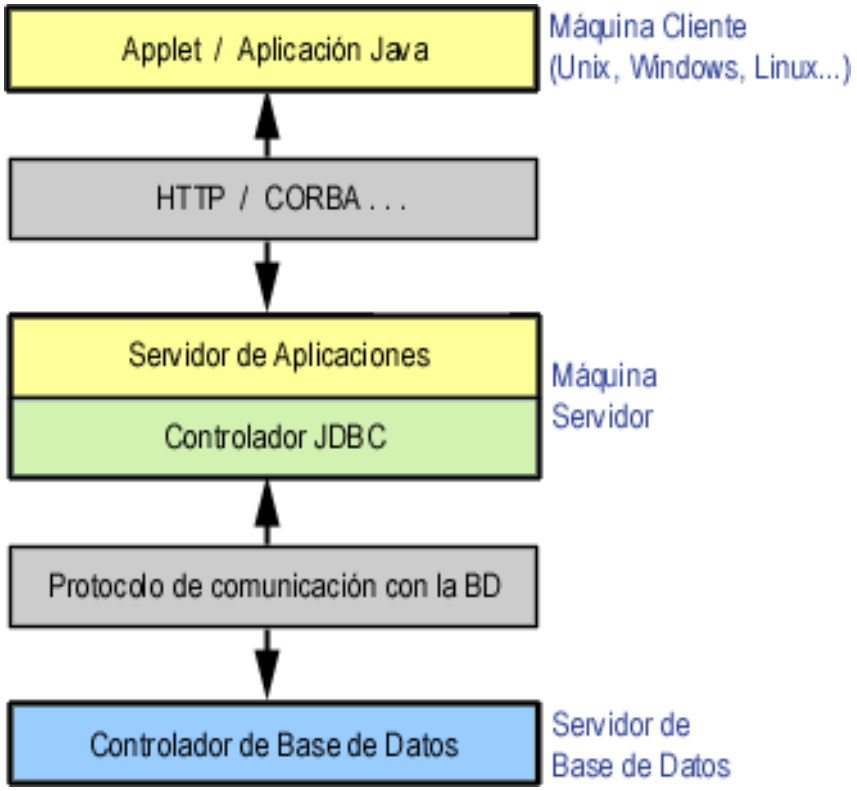
# Arquitecturas de Acceso a datos

- Modelo de dos capas:
  - La aplicación Java se conecta directamente a la base de datos.



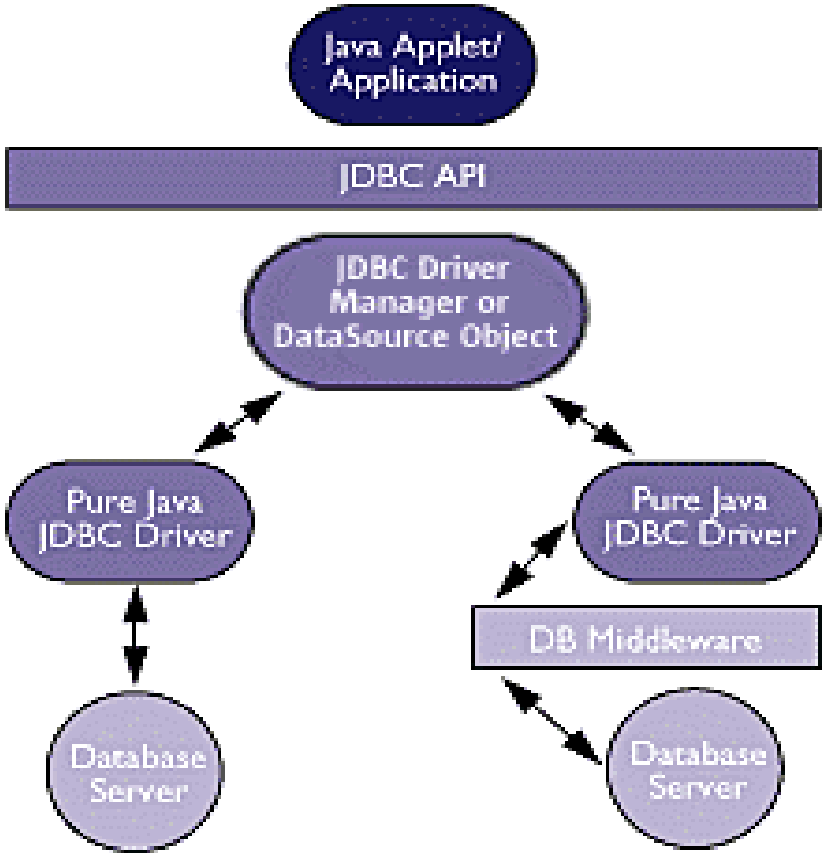
# Arquitecturas de Acceso a datos

- Modelo de tres capas:
  - Capa intermedia que encapsula el acceso a la base de datos.

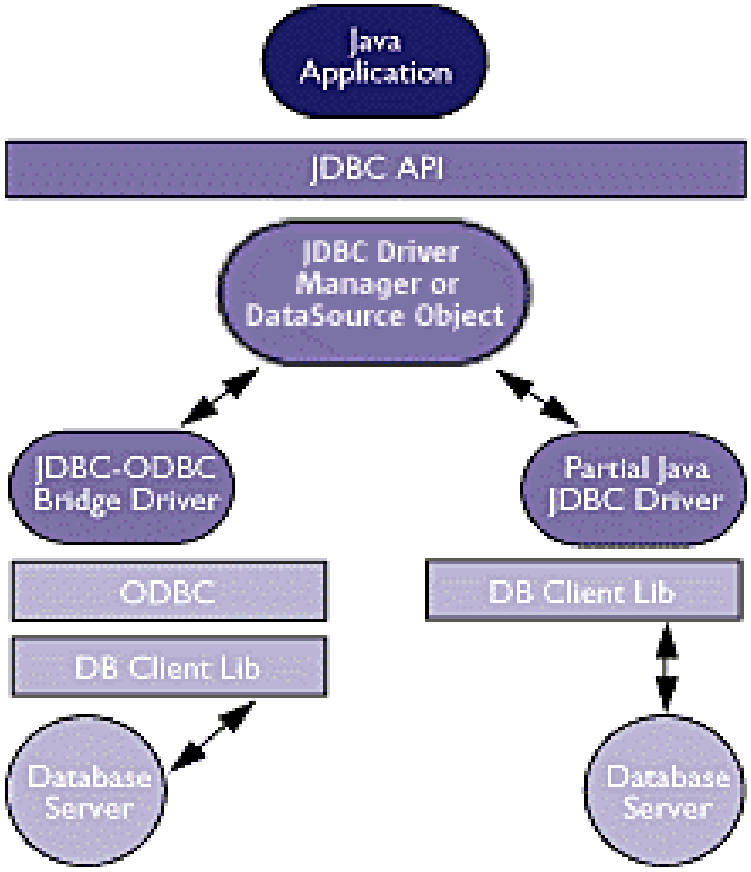


# Tipos de Drivers JDBC

- **Direct-to-Database Pure Java Driver**
  - Convierte las llamadas JDBC al protocolo de RED usado directamente por el DBMS.
- Pure Java Driver for Database Middleware
  - En lugar de comunicarse directamente con la base de datos se comunica con un componente de la DB que traduce las sentencias.



- **JDBC-ODBC Bridge + ODBC Driver**
  - Convierte las llamadas JDBC al protocolo de ODBC .
- **API Java Parcial**
  - Este tipo de driver convierte las llamadas JDBC en llamadas al API del cliente para Oracle, DB2, u otra.
  - Requiere tener algún tipo de código binario del lado del cliente.



# SQL – Structured Query Language

- Lenguaje estándar para acceso a bases de datos relacionales (ANSI SQL)
- Problemas:
  - No existe una implementación única, y cada DBMS extiende el SQL estándar de manera propietaria.
  - Los tipos de datos pueden variar según el DBMS empleado.
- Con JDBC logramos independencia de la plataforma:
  - Implementa un contenedor de tipos de datos genéricos (java.sql.Types)
  - Permite el envío de sentencias SQL propietarias directamente a través del driver JDBC apropiado.

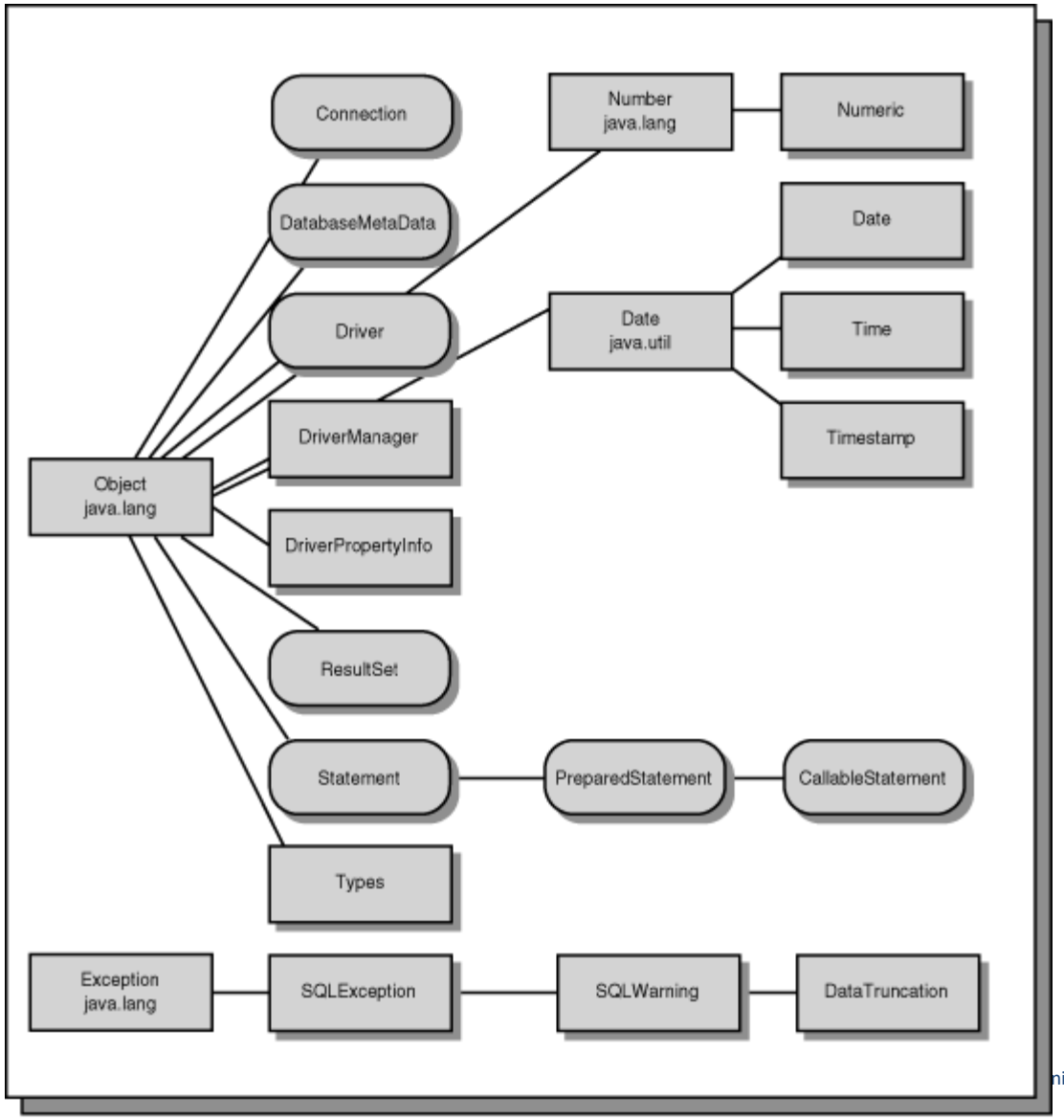
# API JDBC

- java.sql
  - Funcionalidad básica JDBC (lado cliente)
- javax.sql (JDBC Optional Package)
  - Extiende la funcionalidad de las APIs JDBC hacia el lado de servidor.
  - Forma parte de JEE.
- Versiones JDBC:
  - API JDBC 4.0 → incluida a partir de JSE versión 1.6
  - API JDBC 3.0 → incluida a partir de JSE versión 1.4
  - API JDBC 2.0 → incluida a partir de JSE versión 1.2
  - API JDBC 1.0 → incluida a partir de JSE versión 1.1

# Versiones de JDBC

- JDBC 1.0
  - Provee el framework básico para acceso a datos.
- JDBC 2.0
  - Núcleo (java.sql) → intenta mejorar la performance y funcionalidad (ejemplos: scrollable result sets).
  - Nuevos tipos de datos.
  - Conjunto de clases que forman un paquete opcional (javax.sql)
- JDBC 3.0
  - Agrega nuevas características a la API de metadatos
  - Recuperación de claves auto generadas, entre otras mejoras
- JDBC 4.0
  - Administración automática del driver.
  - Mejoras en la administración de conexiones
  - Uso de anotaciones para especificar consultas SQL.

# API JDBC


Connection
Number
java.lang
Numeric
DatabaseMetaData
Date
Driver
Date
java.util
Time
Timestamp
DriverManager
Object
java.lang
DriverPropertyInfo
ResultSet
Statement
PreparedStatement
CallableStatement
Types
Exception
java.lang
SQLException
SQLWarning
DataTruncation

# El Paquete java.sql

| Clase / Interface | Descripción |
|---|---|
| Driver | Permite conectarse a una base de datos: cada DBMS requiere un driver Distinto. |
| DriverManager | Permite gestionar todos los drivers instalados en el sistema. |
| DriverPropertyInfo | Representa una propiedad utilizada por el driver para establecer una conexión. Cada driver utiliza distintas propiedades. |
| Connection | Representa una conexión con una base de datos. Una aplicación puede tener más de una conexión a más de una base de datos. |
| DatabaseMetadata | Proporciona información acerca de una Base de Datos, como las tablas que contiene, etc. |

# El Paquete java.sql

| Clase / Interface | Descripcion |
|---|---|
| Statement | Ejecuta sentencias SQL sin parámetros. |
| PreparedStatement | Ejecuta sentencias SQL con parámetros de entrada. |
| CallableStatement | Ejecuta sentencias SQL con parámetros de entrada/salida (procedimientos almacenados). |
| ResultSet | Recupera registros obtenidos al ejecutar una sentencia SELECT. |
| ResultSetMetadata | Obtiene información sobre un ResultSet: cantidad de columnas, sus nombres, etc. |
| ParameterMetaData | Información sobre los tipos y las propiedades de los parámetros de un PreparedStatement. |

# Primeros pasos con JDBC

1. Instalar Java y el JDBC (Junto con el JDK también viene JDBC)

2. Instalar un driver.

   Para los drivers JDBC escritos para controladores de bases de datos específicos, la instalación consiste sólo en copiar el driver (no se necesita ninguna configuración especial).

http://www.oracle.com/technetwork/java/index-136695.html

3. Instalar y configurar la base de datos

# Requisitos para el uso de JDBC

1. Adquirir un driver JDBC para la base de datos a utilizar.

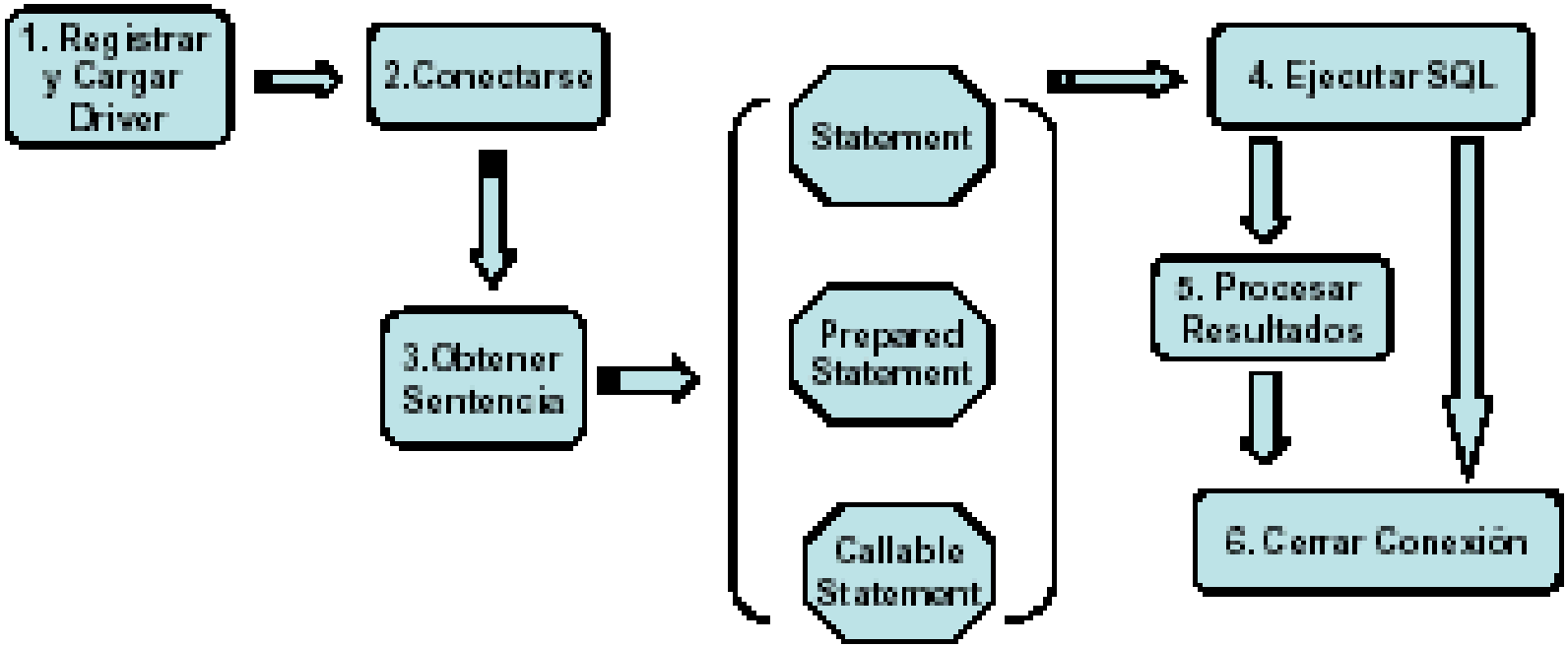
2. Incluir el jar que posee el driver en el classpath de nuestra aplicación:

   – set classpath=…;c:\drivers\jdbc_dataDrivers.jar

3. Importar en el código fuente las clases JDBC.

# Uso de JDBC

- Se debe seguir una secuencia ordenada de pasos para acceder a la base de datos, recuperar información y trabajar con ella:



# Paso 1 - Registrar y cargar el driver

- **`DriverManager`** → clase que se encarga de cargar inicialmente todos los drivers JDBC disponibles.
- Método **`forName()`** de la clase **`Class`** → se encarga de cargar en memoria la clase del driver.
- Todas las clases driver tienen una sección estática que crea una instancia del driver y lo registra con **`DriverManager`**. No es necesario crear un driver y registrarlo con el método **`registerDriver`** de la clase **`DriverManager`**.
- Ejemplo: sentencia para crear y registra una instancia del driver para conectar a una base de datos MySQL:

```
Class.forName("org.gjt.mm.mysql.Driver");
```

# Paso 2 - Conectarse

- Una vez cargado el driver → hacer una conexión.
- Se obtiene un objeto del tipo **`Connection`**.
- La conexión se especifica siguiendo una sintaxis basada en una especificación de URL que depende del driver empleado:

  **`"jdbc:subprotocolo//servidor:puerto/db"`**

- Ejemplo de conexión a MySQL:

```
String url =
    "jdbc:mysql://localhost:3306/myDataBase";
Connection con =
    DriverManager.getConnection(url, "usr", "pwd");
```

# Paso 3 - Obtener una sentencia

- Existen clases que permiten enviar sentencias SQL al controlador de la BD:
  - `Statement`
  - `PreparedStament`
  - `CallableStatement`
- Las instancias de estas clases se obtienen invocando al método apropiado de un objeto del tipo **`Connection`**.
- Ejemplo:

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

# Paso 4 - Ejecutar la sentencia

- Interfaz **`Statement`**
  - Usar **`executeQuery(String sql)`** para sentencias SELECT
    - Retorna un objeto `ResultSet` para el procesamiento de las filas.
  - Usar **`executeUpdate(String sql)`** para sentencias DML/DDL
    - Retorna un `int` con el número de filas afectadas.
  - Usar **`execute(String`**) para una sentencia SQL arbitraria
    - Retorna un valor boolean.
- Ejemplos:

```
Statement stmt = con.createStatement();
stmt.executeUpdate("Delete from table1");
stmt.executeQuery("Select * from table1");
```

# Paso 4 - Ejecutar la sentencia

- Interfaz **`PreparedStatement`**
  - A diferencia de **`Statement`**, se proporciona una sentencia SQL en el momento de su creación.
  - Permite utilizar la misma consulta con diferentes parámetros múltiples veces.

- Ejemplo:

```
String sql1 = "Select col1 from t1 where id = ?";
PreparedStatement pStmt = con.prepareStatement(sql1);
pStmt.setInteger(1, 999);
pStmt.executeQuery();
```

# Paso 4 - Ejecutar la sentencia

- Interfaz **`CallableStatement`**
  - Permite utilizar funciones y procedimientos almacenados implementadas directamente sobre el DBMS.
  - Apropiado para:
    - Funciones muy complicadas.
    - Funciones que se ejecutadan varias veces a lo largo del tiempo de vida de la aplicación.
- Ejemplos:

```
String sql2 = "? := FUNCION1(?);"
CallableStatement cStmt = con.prepareCall(sql2);
cStmt.registerOutParameter(1, INTEGER);
cStmt.setInteger(2, 1234);
cStmt.execute();
cStmt.getInt(1);
```

# Paso 5 - Procesar el resultado

- JDBC devuelve los resultados de una consulta SQL (SELECT) en un objeto **`ResultSet`**.
- Objeto **`ResultSet`** → mantiene un cursor apuntando a la fila actual de datos.
- Método **`next()`** → mueve el cursor a la siguiente fila y hace que esa fila (llamada fila actual) sea aquella con la que podemos operar.
- Desde JDBC 2.0 → también se puede mover el cursor hacia atrás, hacia posiciones específicas y a posiciones relativas a la fila actual.
- Los métodos **`getXxx`** del tipo apropiado se utilizan para recuperar el valor de cada columna.
- Ejemplo: para recuperar un valor VARCHAR → **`getString(int i)`**

# Paso 6 - Liberar los recursos

- Para liberar recursos que ya no se necesitan, deben cerrarse explícitamente los objetos:
  - `Connection`
  - `Statement`
  - `ResultSet`

- Invocar a su respectivo método `close()`.

# Ejemplo completo

```
String sql = "SELECT col1, col2 FROM tabla";
String url = "jdbc:mysql://localhost:3306/myDataBase";
try {
  Class.forName("org.gjt.mm.mysql.Driver");
  Connection con = DriverManager.getConnection(
      url, "usr", "pwd");
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery(sql);
  while(rs.next()){
    System.out.printl("Columna 1: " + rs.getString(1));
    System.out.printl("Columna 2: " + rs.getInt("col2"));
  }
} catch(SQLException e) { /* ... */ }
```

# Ejecutar consulta con parámetros

```
String sql = "SELECT col1, col2 FROM tabla WHERE id = ?";
String url = "jdbc:mysql://localhost:3306/myDataBase";
try {
  Class.forName("org.gjt.mm.mysql.Driver");
  Connection con = DriverManager.getConnection(
      url, "usr", "pwd");
  PreparedStatement stmt = con.prepareStatement(sql);
  stmt.setInt(1, 999);
  ResultSet rs = stmt.executeQuery(sql);
  while(rs.next()){
    System.out.printl("Columna 1: " + rs.getString(1);
    System.out.printl("Columna 2: " + rs.getInt("col2");
  }
} catch(SQLException e) { /* ... */ }
```

# Sentencia de actualización de datos

- La actualización, eliminación o inserción de datos se puede realizar de dos formas:

1. Ejecutando las sentencias SQL correspondientes (INSERT – UPDATE – DELETE).

2. Un **`ResultSet`** puede actualizar los valores de una fila, eliminar la fila o insertar una nueva fila.

Para esto: pasar al método **`createStatement()`** del objeto **`Connection`**, la constante **`CONCUR_UPDATABLE`**.

Si se desea que el **`ResultSet`** no se pueda actualizar se debe enviar la constante **`CONCUR_READ_ONLY`**.

# Ejemplo: consulta de actualización de datos

```
String sql = "UPDATE tabla SET " +
    "col1 = 'aa', col2 = col2 * 1.1";
String u = "jdbc:mysql://localhost:3306/myDataBase";
try {
  Class.forName("com.mysql.Driver");
  Connection c =
      DriverManager.getConnection(u, "usr", "pwd");
  Statement stmt = c.createStatement();
  stmt.executeUpdate(sql);
  c.close();
} catch(SQLException e) { /* ... */ }
```

# Ejemplo: ResultSet actualizable

```
Statement stmt =
    c.createStatement(ResultSet.CONCUR_UPDATABLE);
Resultset rs = stmt.executeQuery(sql);
while(rs.next()) {
  rs.updateString(1, "aa");
  rs.updateInt("col2", rs.getInt("col2") * 10);
  rs.updateRow();
}
resultSet.moveToInsertRow();
resultSet.updateString("col1", "aaaa");
resultSet.insertRow();
```

# Obtener metadatos

- El objeto **`ResultSet`** no proporciona información referente a la estructura de cada columna.
- Para ello se utiliza **`ResultSetMetaData`** → que permite obtener...
  - el tipo de cada columna,
  - su nombre,
  - si es del tipo autoincremento,
  - si es sensible a mayúsculas,
  - si se puede escribir en dicha columna,
  - si admite valores nulos,
  - etc.

- Para obtener un objeto de tipo **`ResultSetMetaData`** basta con llamar al método **`getMetaData()`** del objeto **`ResultSet`**.

# Trabajar con metadatos

- Principales Métodos de **`ResultSetMetaData`** :
  - `getTableName()` → Nombre de la tabla a que pertenece la columna.
  - `getColumnCount()` → Número de columnas en el ResultSet.
  - `getColumnName(int)` → Nombre de la columna.
  - `getColumnType(int)` → Tipo de la columna (uno de los valores constantes en `java.sql.Types`)
  - `getColumnClassName(int)` → Nombre de la clase Java de los objetos devueltos como valores de la columna.
  - `isNullable()` → Indica si la columna puede contener un NULL SQL.
  - `isAutoIncrement()` → Indica si la columna es de tipo autoincremento.
  - `isNullable()` → Indica si la columna puede contener un NULL SQL.
  - `isSearchable()` → Indica si la columna puede usarse en una cláusula where.
  - `getColumnDisplaySize()` → Ancho máximo en caracteres necesario para mostrar el contenido de la columna.

# Ejemplo de obtención de metadatos

```
String sql = "SELECT a, b, c FROM tabla";
ResultSet rs = stmt.executeQuery(sql);
ResultSetMetaData rsmd = rs.getMetaData();
int numberOfColumns = rsmd.getColumnCount();
for (int i = 1; i <= numberOfColumns; i++) {
    boolean b = rsmd.isSearchable(i);
    String nombreCol = getColumnName(i);
    // ...
}
```

# Transacciones

- Una transacción es un conjunto de sentencias SQL que deben ser ejecutadas y donde la transacción tiene éxito si todas y cada una de dichas tareas son ejecutadas exitosamente

- Si falla al menos una se deben anular todas las demás sentencias

- Por defecto, una conexión funciona en modo *autocommit*, es decir, cada vez que se ejecuta una sentencia SQL se abre y se cierra automáticamente una transacción, que sólo afecta a dicha sentencia

- Si no se está trabajando en modo autocommit será necesario que se cierren explícitamente las transacciones mediante commit() si tienen éxito, o rollback(), si fallan

# Transacciones

- Con **`setTransactionIsolation()`** se especifica el nivel de aislamiento de una transacción:
  - TRANSACTION_NONE: No se soportan transacciones.
  - TRANSACTION_READ_UNCOMMITTED: pueden ocurrir “dirty reads”, “non-repeatable reads” y “phantom reads”.
  - TRANSACTION_READ_COMMITTED: Se previene que ocurran “dirty reads”.
  - TRANSACTION_REPEATABLE_READ: Se previene que ocurran “dirty reads” y “non-repeatable reads”.
  - TRANSACTION_SERIALIZABLE: Se previenen todos los problemas anteriores.

# Ejemplo de transacciones

```
String sql1 = "UPDATE tablaMaestro SET col2 = col2 * 1.1";
String sql2 = "UPDATE tablaDetalle SET col2 = col2 * 1.1";
try{
  Connection c = DriverManager.getConnection(u, "usr", "pwd");
  c.setAutoCommit(false);
  Statement stmt = c.createStatement();
  stmt.executeUpdate(sql1);
  stmt.executeUpdate(sql2);
  c.commit();
} catch(SQLException e) {
  c.rollback();
}
```

UTN SF DISTANCIA

# Resumen

- Introducción a JDBC
- Arquitectura de Acceso a Datos
- Tipos de drivers
  - Puente JDBC a ODBC
  - JAVA/Binario o Java/Codigo Nativo
  - Java/Protocolo nativo
  - JDBC/Net
- API JDBC
- Requisitos para el uso de JDBC
- Cómo acceder a la base de datos
- Recuperar Resultados
- Transacciones

UTN SF DISTANCIA

# FIN