

# Teórico 9

## APIs para Acceso a Bases de Datos



# Arquitecturas de Aplicaciones con Accesos a Base de Datos

Básicamente tenemos dos Arquitecturas:

- Dos Capas

- La aplicación cliente se conecta al servidor de DB directamente.

- Tres Capas

- La aplicación cliente se conecta al servidor de DB a través de un servidor de aplicaciones.

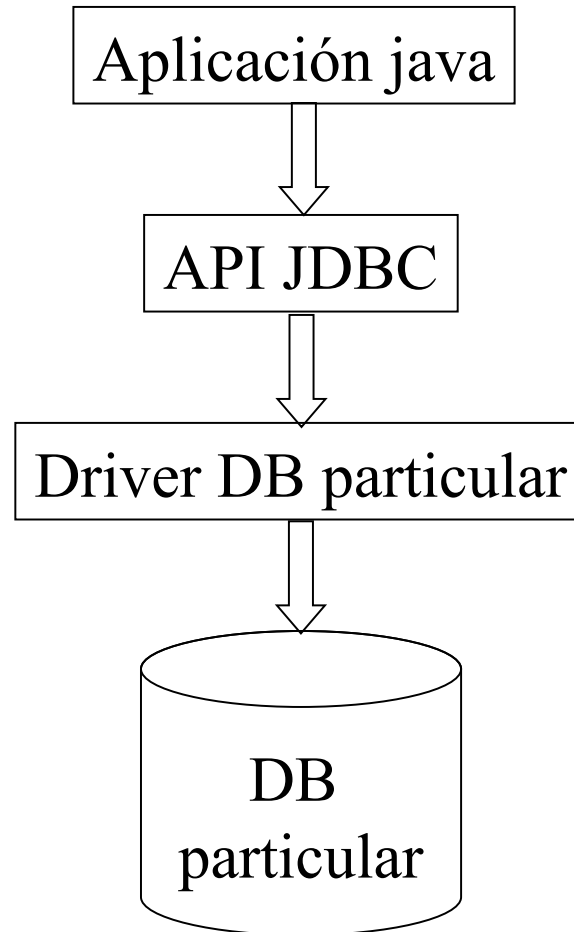
# *Java DataBase Connectivity (JDBC)*

Es una API JAVA que permite la ejecución de operaciones sobre base de datos, independientemente de la base de datos a la cual se accede, utilizando SQL de la base de datos base de datos que se utilice.

Principales características de JDBC:

- Interacción con la DB a través de SQL.
- 100% Java.
- Muy simple de utilizar.
- Alta performance.
- Se puede utilizar cualquier base de datos para la cual este implementado el driver de acceso, por ej.: MySQL, firebird, postgres, Oracle, etc.

# Arquitectura JDBC



# Algunos Componentes de la API

- Clase estática DriverManager
  - Carga el driver de la base de datos elegida.
- Interface Connection
  - Representa una conexión a la base de datos.
- Interface Statement y subclases
  - Representa una sentencia SQL.
- Interface ResultSet
  - Para tratar registros retornados por la ejecución de un Statement.
- Interface DatabaseMetaData
  - Posee métodos para extraer información sobre los metadatos, por ejemplo, tablas , atributos de las tablas, etc.

# URL JDBC

- Una URL JDBC especifica la base se datos a conectarse y se compone de la siguiente forma:

`jdbc:subprotocolo:fuelle`

- Cada Driver tiene su propio subprotocolo.
- Cada subprotocolo tiene su propia sintaxis para la fuente.



# Ejemplos de URLs JDBC

- PostgreSQL

`jdbc:postgresql://host[:port]/database`

Ejemplo:

`jdbc:postgresql://localhost:5432/prueba`

- Mysql

`jdbc:mysql://host[:port]/database`

Ejemplo:

`jdbc:mysql://localhost:3306/ejercicio1`

# Clase DriverManager

- Provee el método

`getConnection(String url, String user, String password)`

Este método retorna un objeto que implementa la Interface Connection que representa una conexión con la Base de Datos.



# Interface Connection

- Representa una sesión con una base de datos particular.
- En la sesión se ejecutan instrucciones SQL y se obtienen los resultados.
- Se puede obtener información de “metadata”(estructura) de la base de datos.
- Provee métodos para manejo de transacciones (commit(), rollback())

# Connection Provee Métodos para la Creación de Statement

La Interface provee métodos para la creación de objetos que implementan las interfaces Statement, PreparedStatement y CallableStatement, que representan una sentencia SQL a ejecutar en el motor de bases de datos.

# Método setAutoCommit

setAutoCommit(boolean)

- Si es Verdadero, cada ejecución de una sentencia SQL se hace es una transacción.
- Si AutoCommit es Falso, se debe utilizar explícitamente los métodos commit o rollback para delimitar la transacción.

# Ejemplo para obtener una conexión a una base de datos MySQL

```
String driver = "org.gjt.mm.mysql.Driver";  
String url = "jdbc:mysql://localhost/prueba";  
String username = "root";  
String password = "root";  
  
// carga el driver de la base de datos si no se cargo.  
Class.forName(driver);  
  
// establece la conexión de red a la base de datos.  
Connection connection =  
DriverManager.getConnection(url, username, password);
```

# Interface Statement

- Representa una sentencia SQL estática, es decir, no se le puede pasar parámetros.

# Interface PreparedStatement

- Hereda de Statement y representa un SQL precompilado, permite el pasaje de parámetros.

# Interface CallableStatement

- Hereda de Statement, su objetivo es la ejecución de procedimientos almacenados de la base de datos.

# Métodos de la Interface Connection para la Creación de Statements

La Interface Connection provee métodos para crear diferentes tipos de Statement:

**Statement createStatement()**

- Retorna un nuevo objeto Statement.

**PreparedStatement prepareStatement(String sql)**

- Retorna un nuevo objeto PreparedStatement.

**CallableStatement prepareCall(String sql)**

- Retorna un nuevo objeto CallableStatement.

- La optimización es el objetivo de tener diferentes tipos de Statement.



# Parámetros para la Creación de Statement

- A la creación de Statement se le puede pasar varios parámetros.
- Un parámetro es el tipo de ResultSet a retornar, puede tomar los siguientes valores:
  - `ResultSet.TYPE_FORWARD_ONLY`: Sólo avanza hacia adelante.
  - `ResultSet.TYPE_SCROLL_INSENSITIVE`: puede ser recorrido en ambos sentidos y las actualizaciones no se reflejan hasta que se haga la consulta de nuevo.
  - `ResultSet.TYPE_SCROLL_SENSITIVE`. puede ser recorrido en ambos sentidos y las actualizaciones se reflejan cuando se producen.

# Parámetros en la creación de Statement (Cont.)

- Otro parámetro es para determinar si el ResultSet es actualizable o no.
- Puede tomar los siguientes valores:
  - `ResultSet.CONCUR_READ_ONLY`: El resultado es de sólo lectura.
  - `ResultSet.CONCUR_UPDATABLE`.: El resultado es Actualizable.

# Algunos Métodos de Statement

`ResultSet executeQuery(String)`

- Ejecuta una sentencia SQL que retorna un `ResultSet`.

`int executeUpdate(String)`

- Ejecuta sentencia SQL del tipo `INSERT`, `UPDATE` o `DELETE`. Retorna el numero de registros afectados por la ejecución de la sentencia.

`boolean execute(String)`

- Ejecuta un sentencia SQL que puede retornar múltiples resultados.

# Interface ResultSet

- Un ResultSet provee acceso a una tabla de datos generada por la ejecución de un sentencia SQL, a través de un Statement.
- Un único ResultSet por Statement puede ser abierto al mismo tiempo.
- Las filas de la tabla se pueden recuperar en secuencia o no, dependiendo de como se creó el Statament que retorna el ResultSet.
- Un ResultSet mantiene un cursor que apunta a la fila o registro actual de la tabla.
- El método next() mueve el cursor a la siguiente fila.

# Métodos de ResultSet

- **boolean next():** Corre el cursor a la próxima fila. La primera llamada next() activa la primer fila. Retorna falso cuando no hay mas filas.
- **void close():** Libera el ResultSet.
- **<Type> get<Type>(int columnIndex):** Retorna el valor(del tipo *Type*) del campo para la fila actual. ColumnIndex comienza de 1. Ejemplo: getString(1), retorna el valor String de la columna 1 del ResultSet.
- **<Type> get<Type>(String columnName):** Ídem anterior pero tiene como parámetro el nombre de la columna. Es menos eficiente que el anterior.
- **update<Type>(String columnLabel, <Type> valor):** Actualiza lo almacenado en la fila actual en la columna *columnLabel* con el valor *valor*.
- **int findColumn(String columnName):** Retorna el índice de la columna con nombre columnName.

# Métodos de ResultSet (Cont.)

- **previous():** mueve el cursor una fila atrás. Retorna true si el cursor se pudo posicionar, sino false.
- **first():** mueve el cursor a la primera fila. Retorna true si el cursor se posicionó en la primera fila, retorna false si el ResultSet no tiene filas.
- **last():** mueve el cursor a la última fila. Retorna true si el cursor se posicionó en la última fila, retorna false si el ResultSet es vacío.
- **beforeFirst():** posiciona el cursor al comienzo del ResultSet, antes de la primera fila. Si el ResultSet es vacío el método no tiene efecto.
- **afterLast():** posiciona el cursor al final del ResultSet, después de la última fila. Si el ResultSet es vacío el método no tiene efecto.
- **relative(int cant):** mueve el cursor *cant* filas relativas a la posición corriente.
- **absolute(int n):** posiciona el cursor a la fila *n* del ResultSet.

# Ejemplo

```
Statement statement = connection.createStatement();  
String query = "SELECT * FROM persona";
```

```
//Envía en query a la base de datos y almacena el resultado.
```

```
ResultSet resultSet = statement.executeQuery(query);
```

```
// Muestra los resultados.
```

```
while(resultSet.next())  
{  
    System.out.print(" DNI: " + resultSet.getString("DNI"));  
    System.out.print("; Nombre: "+resultSet.getString("nombre"));  
    System.out.print("; Email: " + resultSet.getString("email")) ;  
    System.out.print("\n    ");  
    System.out.print("\n    ");  
}
```

# Correspondencia de tipos SQL y JAVA

## Tipos SQL

CHAR, VARCHAR, LONGVARCHAR  
NUMERIC, DECIMAL  
BIT  
TINYINT  
SMALLINT  
INTEGER  
BIGINT  
REAL  
FLOAT, DOUBLE  
BINARY, VARBINARY, LONGVARBINARY  
DATE  
TIME  
TIMESTAMP

## Tipos Java

String  
java.math.BigDecimal  
boolean  
byte  
short  
int  
long  
float  
double  
byte[]  
java.sql.Date  
java.sql.Time  
java.sql.Timestamp