**BASE DE DATOS**

**El lenguaje SQL**

**Qué es SQL**

El Lenguaje de Consulta Estructurado (SQL=Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar información de interés de una base de datos, de una forma sencilla.

El SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones sobre los mismos. Es un lenguaje declarativo de alto nivel, que, gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación.

Es el lenguaje utilizado universalmente para interactuar con base de datos, permitiendo realizar consultas, inserciones, actualizaciones y eliminaciones de datos, como así también de base de datos.

**Dónde se utiliza**

En la actualidad el SQL es el estándar de facto de la inmensa mayoría de los Administradores de Base de Datos comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy amplio. El soporte estándar se denomina ANSI SQL.

Entre los sistemas de gestión de base de datos con soporte SQL más utilizados se encuentran los siguientes:

* DB2
* Oracle
* SQL Server
* MySQL
* PostgreSQL
* Informix

**MySQL como Data Base Management System**

**Introducción**

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario. Inicialmente, carecía de elementos considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones. A pesar de ello, atrajo a los desarrolladores de páginas web con contenido dinámico, justamente por su simplicidad y velocidad.

Poco a poco los elementos de los que carecía MySQL están siendo incorporados tanto por desarrollos internos, como por desarrolladores de software libre.

**Características**

Entre las características disponibles en las últimas versiones se puede destacar:

* Amplio subconjunto del lenguaje SQL. Algunas extensiones son incluidas igualmente.
* Disponibilidad en gran cantidad de plataformas y sistemas.
* Diferentes opciones de almacenamiento según si se desea velocidad en las operaciones o el mayor número de operaciones disponibles.
* Transacciones y claves foráneas.
* Conectividad segura.
* Replicación.
* Búsqueda e indexación de campos de texto.

La licencia GNU GPL de MySQL obliga a distribuir cualquier producto derivado (aplicación) bajo esa misma licencia. Si un desarrollador desea incorporar MySQL en su producto, pero no desea distribuirlo bajo licencia GNU GPL, puede adquirir la licencia comercial de MySQL que le permite hacer justamente eso.

**¿Qué es DDL?**

El lenguaje de Definición de datos, en inglés Data Definition Language (DDL), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Algunas de sus operaciones básicas son: CREATE, ALTER y DROP.

Involucra los comandos necesarios para crear, modificar y eliminar una tabla y una base de datos, como así también permite crear claves primarias, índices y restricciones.

**La operación CREATE**

Este comando crea un objeto dentro de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte.

A continuación, un ejemplo de utilización para la creación de una tabla:

CREATE TABLE TABLA\_NOMBRE (

 my\_field1   INT UNSIGNED,

 my\_field2   VARCHAR (50),

 my\_field3   DATE NOT NULL,

 PRIMARY KEY (my\_field1, my\_field2)

)

**La operación ALTER**

Este comando permite modificar la estructura de un objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, modificar un trigger, etc.

A continuación, un ejemplo de utilización para agregar una columna a una tabla:

ALTER TABLE TABLA\_NOMBRE (

 ADD NUEVO\_CAMPO INT UNSIGNED

)

**La operacion DROP**

Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte. Se puede combinar con la sentencia ALTER.

A continuación, un ejemplo de utilización para agregar eliminar una tabla:

DROP TABLE TABLA\_NOMBRE

**¿Qué es DML?**

DML significa Data Manipulation Language o Lenguaje de Manipulación de Datos, y corresponde las sentencias del SQL que se utilizan para manejar los datos de la base de datos (select, insert, update, delete, etc).

Involucra los comandos necesarios para hacer consultas, inserciones, modificaciones y eliminaciones. Dichos comandos están presentados de la siguiente manera:

* SELECT – obtiene información de una base de datos
* INSERT INTO – inserta información en una base de datos
* UPDATE - actualiza información de una base de datos
* DELETE – elimina información de una base de datos

**El comando SELECT**

El comando SELECT se utiliza para seleccionar información de una tabla. Para seleccionar todas las columnas se utiliza el \* (asterisco), y la cláusula WHERE se utiliza para establecer un criterio de búsqueda.

Para traer todos los datos de una tabla, se realiza la siguiente consulta:

SELECT \* FROM tabla\_nombre

Si es necesario traer únicamente los datos de una columna, se realiza de la siguiente manera:

SELECT nombre\_columna(s) FROM tabla\_nombre

Para traer los registros según una condición, se realiza de la siguiente manera:

SELECT nombre\_columna(s) FROM tabla\_nombre WHERE campo1 = valor1

**El comando INSERT**

El comando INSERT se utiliza para insertar datos en una tabla. La inserción se realiza de la siguiente manera:

INSERT INTO "nombre\_tabla" ("columna1", "columna2", ...)

VALUES ("valor1", "valor2", ...)

Se pueden agregar datos en grupo o especificando la columna donde es necesario ingresar la información. Para agregar información a todos los campos, el uso es el siguiente:

INSERT INTO nombre\_tabla

VALUES (valor1, valor2,....)

Para agregar valores en cada columna de manera particular, se realiza de la siguiente forma:

INSERT INTO nombre\_tabla(columna1, columna2, ....)

VALUES (valor1, valor2,....)

**El comando UPDATE**

El comando UPDATE se utiliza para actualizar registros, y su forma de utilización es la siguiente:

UPDATE nombre\_tabla SET nombre\_columna1 = nuevo\_valor,

nombre\_columna2 = otro\_valor

WHERE nombre\_columna = algun\_valor

**El comando DELETE**

El comando DELETE se utiliza para eliminar registros, su forma de utilización es la siguiente:

DELETE FROM nombre\_tabla WHERE nombre\_columna = algun\_valor

# JDBC: CONEXIÓN CON BASE DE DATOS

## Introducción

### Qué es JDBC

JDBC es el acrónimo de Java Database Connectivity, una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede utilizando el lenguaje SQL del modelo de base de datos que se utilice.

Está conformada por diversas clases e interfaces ubicadas en el paquete java.sql.

### La necesidad de una librería

Al trabajar con JDBC resulta necesario agregar un jar al proyecto que contiene las clases necesarias que se utilizan para “dialogar” con un DBMS. Cada DBMS tiene su propio archivo jar. Estos archivos se pueden obtener de:

[http://developers.sun.com/product/jdbc/drivers](https://www.google.com/url?q=http://developers.sun.com/product/jdbc/drivers&sa=D&ust=1503407313684000&usg=AFQjCNH6cZt0Nz3mDSGqg8WJGSO40YbSNA)

También es posible conseguir estos archivos jar en la página web correspondiente a cada DBMS, por ejemplo, en caso de usar un DMBS Oracle es posible buscarlo en la página de Oracle:[www.oracle.com](https://www.google.com/url?q=http://www.oracle.com/&sa=D&ust=1503407313684000&usg=AFQjCNGCRRhW5jUMx1es_4TEKIA6rxw9dA)

## Conexión con la base de datos

### La interfaz Connection

Para poder trabajar con una base de datos, el punto de partida siempre es conseguir una conexión, es decir un objeto del tipo Connection (este objeto pertenece a una clase que implementa la interfaz Connection).

### Construcción de un Administrador de Conexiones

Para poder obtener una conexión, una forma simple y cómoda de trabajar es armar una clase llamada, por ejemplo, AdministradorDeConexiones, que contenga dentro de un método obtenerConexion() el código necesaria para obtenerla. A continuación, se presenta un ejemplo de la clase con su correspondiente método:

package ar.com.educacionit.database;

import java.sql.Connection;

import java.sql.DriverManager;

public abstract class AdministradorDeConexiones {

public static Connection obtenerConexion() throws Exception

{

        // Establece el nombre del driver a utilizar

        String dbDriver = "com.mysql.jdbc.Driver";

            // Establece la conexion a utilizar

        String dbConnString = "jdbc:mysql://localhost/baseDatos";

            // Establece el usuario de la base de datos

        String dbUser = "root";

        // Establece la contraseña de la base de datos

        String dbPassword = "";

        // Establece el driver de conexion

        Class.forName(dbDriver).newInstance();

        // Retorna la conexion

        return DriverManager.getConnection(dbConnString, dbUser, dbPassword);

        }

}

## Cómo consultar datos

### El metodo createStatement()

El método createStatement() se utiliza para crear un objeto que modela a una sentencia SQL. Es un objeto del tipo de una clase que implementa la interfaz Statement, y provee la infraestructura para ejecutar sentencias SQL sobre una conexión con una base de datos.

La forma de construir un objeto de este tipo es:

Statement stmtConsulta = laConexion.createStatement();

### El metodo executeQuery()

El método executeQuery() se utiliza para ejecutar una sentencia SQL y obtener el resultado correspondiente dentro de un objeto del tipo ResultSet. Este objeto representa un conjunto de resultados que se obtienen como consecuencia de ejecutar la sentencia SQL del tipo SELECT a traves de la conexión.

La forma de generar un objeto de este tipo es:

ResultSet rs = stmtConsulta.executeQuery(laConsulta);

### Como realizar una consulta

// Define la conexion

Connection laConexion = AdministradorDeConexiones.getConnection();

// Arma la consulta y la ejecuta

String laConsulta = "SELECT \* FROM alumnos";

Statement stmtConsulta = laConexion.createStatement();

ResultSet rs = stmtConsulta.executeQuery(laConsulta);

// Muestra los datos

while( rs.next() ){

System.out.println( "ID: " + rs.getInt("alu\_id") + " -- " +

"Nombre: " + rs.getString("alu\_nombre") + " -- " + “Apellido: " + rs.getString("alu\_apellido") );

}

// Cierra el Statement y la Connection

stmtConsulta.close();

laConexion.close();

## Cómo insertar datos

### El método createStatement()

El método createStatement() es el mismo presentado en la sección Consulta.

### El método execute()

El método execute() se utiliza para ejecutar sentencias SQL del tipo INSERT, UPDATE o DELETE, y a diferencia del método executeQuery() no retorna un conjunto de resultados.

La forma de utilizar el método execute() es:

String laInsercion =

"INSERT INTO alumnos (alu\_id, alu\_nombre, alu\_apellido) VALUES

(101, 'Manuel', 'Santos')";

stmtInsercion.execute(laInsercion);

### Cómo realizar una inserción

// Define la conexion

Connection laConexion = AdministradorDeConexiones.getConnection();

// Arma la sentencia de inserción y la ejecuta

String laInsercion = "INSERT INTO alumnos (alu\_id, alu\_nombre,

alu\_apellido) VALUES (101, 'Manuel', 'Santos')";

Statement stmtInsercion = laConexion.createStatement();

stmtInsercion.execute(laInsercion);

// Cierra el Statement y la Connection

stmtInsercion.close();

laConexion.close();

// Informa que la inserción ha sido realizada con éxito

System.out.println("La inserción ha sido realizada con éxito...");

## Cómo actualizar datos

### El método createStatement()

El método createStatement() es el mismo presentado en la sección Consulta.

### El método execute()

El método execute() es el mismo presentado en la sección Inserción.

### Cómo realizar una actualización

// Define la conexion

Connection laConexion = AdministradorDeConexiones.getConnection();

// Arma la sentencia de actualización y la ejecuta

String laActualizacion = "UPDATE alumnos SET alu\_apellido =

'Trobbiani' WHERE alu\_id = 101";

Statement stmtActualizacion = laConexion.createStatement();

stmtActualizacion.execute(laActualizacion);

// Cierra el Statement y la Connection

stmtActualizacion.close();

laConexion.close();

// Informa que la actualización ha sido realizada con éxito

System.out.println("La actualización ha sido realizada con

éxito...");

## Cómo eliminar datos

### El método createStatement()

El método createStatement() es el mismo presentado en la sección Consulta.

### El método execute()

El método execute() es el mismo presentado en la sección Inserción.

### Cómo realizar una eliminación

// Define la conexion

Connection laConexion = AdministradorDeConexiones.getConnection();

// Arma la sentencia de eliminación y la ejecuta

String laEliminacion = "DELETE FROM alumnos WHERE alu\_id = 101";

Statement stmtEliminacion = laConexion.createStatement();

stmtEliminacion.execute(laEliminacion);

// Cierra el Statement y la Connection

stmtEliminacion.close();

laConexion.close();

// Informa que la eliminación ha sido realizada con éxito

System.out.println("La eliminación ha sido realizada con

éxito...");

## La Clase PreparedStatement

Este tipo especial de sentencia es un derivado de la clase que ya conocemos Statement.

La diferencia principal al utilizar un PreparedStatement es que, al contrario del objeto Statement, este recibe una sentencia SQL cuando es creado. Como resultado, el objeto PreparedStatement contiene una sentencia SQL que ha sido precompilada, esto significa que cuando el PreparedStatement es ejecutado, el DBMS puede simplemente correr la sentencia SQL sin tener que compilarla primero.

Si bien esta clase puede ser utilizada para ejecutar sentencias sin parámetros, otra de sus grandes ventajas es la posibilidad de elaborar una sentencia que recibe parámetros facilitando la reutilización del código y la utilización de diferentes tipos de datos.

A continuación, observaremos un ejemplo de cómo utilizar la clase PreparedStatement para crear una sentencia que recibe parámetros:

// Define la conexion

Connection laConexion = AdministradorDeConexiones.getConnection();

// Arma la sentencia de inserción y la ejecuta

String laInsercion = "INSERT INTO alumnos (alu\_id, alu\_nombre,

alu\_apellido) VALUES (?,?,?)";

PreparedStatement stmtInsercion = laConexion.prepareStatement(lainsercion);

// La cantidad de parametros debe ser igual a la cantidad de signos de pregunta en el string predefinido para la sentencia.

stmtInsercion.setInt(1,101); // El primer parámetro de este método es la posición en la que el valor debe ser ubicado.

stmtInsercion.setString(2,”Manuel”);

stmtInsercion.setString(3,”Santos”);

// Hay un metodo set para cada tipo de dato.

stmtInsercion.execute();

// Cierra el Statement y la Connection

stmtInsercion.close();

laConexion.close();

// Informa que la inserción ha sido realizada con éxito

System.out.println("La inserción ha sido realizada con éxito...");

## Transacciones

## Qué es un DAO

Cuando se pretende modelar con objetos un modelo de datos, es decir las tablas y sus relaciones, es muy común utilizar una propuesta simple que consiste en armar una clase por cada tabla que existente.

Dicha clase tendrá los métodos de acceso a la tabla correspondiente, entre ellos la inserción, modificación y la eliminación. Este tipo de clases suelen conocerse como DAOs, ya que objetos de esta clase se utilizarán para realizar operaciones de datos.

De esta forma, si existe la tabla autos, es posible construir una clase denominada Auto, que, siguiendo la especificación de un DAO, debería tener los métodos insertar, modificar, eliminar, y métodos con las consultas que resulten necesarias.

## Qué es una transacción

Una transacción en SQL es una colección de sentencias DML que forman una unidad lógica de trabajo o procesamiento, con propiedades bien definidas. De esta manera será un conjunto de operaciones sobre los datos en una base de datos que o se ejecuta entera o no se ejecuta ninguna de sus sentencias.

JDBC permite que las declaraciones de SQL sean agrupadas juntas en una sola transacción. De esta manera, es posible asegurar la atomicidad y consistencia de datos, usando características transaccionales de JDBC.

## El método setAutoCommit()

El control de la transacción es realizado por el objeto de la conexión. Cuando se crea una conexión, por defecto está en el modo activado. Esto significa que cada declaración individual del SQL es tratada como transacción por sí mismo, y será comprometida tan pronto como finalice la ejecución.

El método setAutoCommit es el encargado de establecer si se trabajara agrupando varias sentencias en una transacción, o por el contrario cada sentencia SQL será una transacción independiente.

Para trabajar con varias sentencias SQL y ejecutarlas como una transacción es necesario establecer el auto-commit en false:

      unaConexion.setAutoCommit(false);

Si no es necesario trabajar con transacciones, será necesario establecer el auto-commit en true:

      unaConexion.setAutoCommit(true);

Por defecto el auto-commit esta seteado en true.

### El método commit()

Cuando el auto-commit está en falso, para poder comprometer o impactar los cambios en la base de datos, es necesario llamar al método commit(). Si no se llama al método commit(), los cambios no serán reflejados en la base de datos aun cuando se hayan ejecutados una o más sentencias SQL.

El método commit() pertenece a la conexión, y la forma de invocarlo es la siguiente:

unaConexion.commit();

### El método rollback()

Si en algún punto antes de invocar el método commit(), el método rollback() es invocado, todas las sentencias que se hayan ejecutado quedarán sin efecto, es decir que el rollback() vuelve atrás todos los cambios realizados sobre los datos desde el último commit() realizado

El método rollback() también pertenece a la conexión, y la forma de invocarlo es la siguiente:

unaConexion.rollback();

### Utilización de transacciones

// Declara la conexion

Connection conn = null;

// Define la conexion

conn = AdministradorDeConexiones.getConnection();

// Setea el auto-commit en falso

conn.setAutoCommit(false);

// Setea el auto-commit en falso

Alumno a1 = new Alumno("Juan");

Alumno a2 = new Alumno("Mario");

Alumno a3 = new Alumno("Pepe");

// Acá comienza la transacción

alumno1.insertar(conn);

alumno2.insertar(conn);

alumno3.insertar(conn);

// Confirma los cambios

conn.commit();

// Cierra la conexion

conn.close()

### Utilización de transacciones con manejo de excepciones

// Declara la conexion

Connection conn = null;

try

{

// Define la conexion

    conn = AdministradorDeConexiones.getConnection();

    // Setea el auto-commit en falso

    conn.setAutoCommit(false);

    // Acá comienza la transacción

    alumno1.insertar(conn);

    alumno2.insertar(conn);

    alumno3.insertar(conn);

// Confirma los cambios

    conn.commit();

}

catch(Exception e)

{

try

    {

              // Vuelve atras los cambios

          conn.rollback();

}

    catch(Exception ee){ // Manejo de errors }

}

finally

{

try

        {

                  // Cierra la conexion

              if( conn != null )  conn.close();

}

    catch(Exception e){ // Manejo de errors }

}