

UD 2: MANEJO DE CONECTORES

ACCESO A DATOS

OBJETIVO DE LA UNIDAD

DESARROLLAR APlicaciones que
GESTIONAN INFORMACIÓN ALMACENADA
EN BASES DE DATOS RELACIONALES
IDENTIFICANDO Y UTILIZANDO
MECANISMOS DE CONEXIÓN

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN
2. MODOS DE ACCESO A BASES DE DATOS
3. BASES DE DATOS EMBEBIDAS
4. PROTOCOLOS DE ACCESO A DATOS
5. JDBC
6. ESTABLECER CONEXIÓN A LA BD
7. ACCESO A METADATOS
8. EJECUCIÓN DE SENTENCIAS
9. GESTIÓN DE TRANSACCIONES

1. INTRODUCCIÓN

LOS **CONECTORES** SON ELEMENTOS QUE SIRVEN PARA:

ACCEDER DESDE UN PROGRAMA (ESCRITO EN EL LENGUAJE DE PROGRAMACIÓN QUE SEA)

A DATOS ALMACENADOS EN UN ORIGEN DE DATOS (QUE PUEDE SER DE DIFERENTE NATURALEZA)

EN ESTA UNIDAD NOS CENTRAREMOS EN LOS ORÍGENES DE DATOS BASADOS EN BASES DE DATOS RELACIONALES

2. MODOS DE ACCESO A BASES DE DATOS

2.1 MODO CONSOLA

2.2 DESDE EL ENTORNO GRÁFICO DEL SGBD

2.3 DESDE CUALQUIER PROGRAMA SOFTWARE

2.1 MODO CONSOLA

SE INTRODUCEN COMANDOS DESDE LA LÍNEA DE COMANDOS DEL PROPIO SISTEMA OPERATIVO

PASOS A SEGUIR:

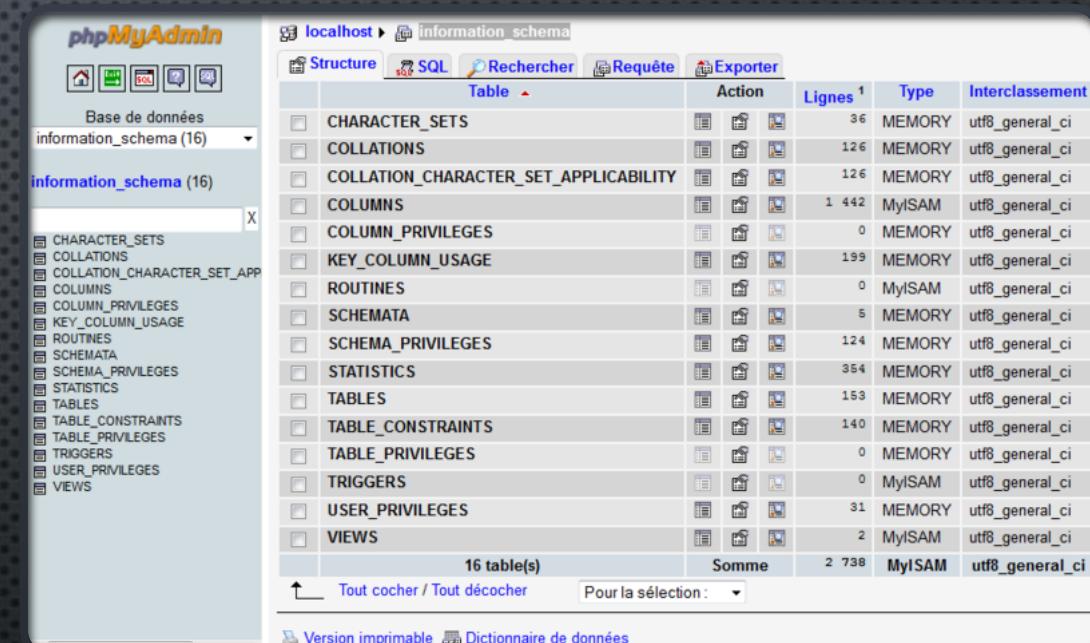
- ARRANCAR EL MOTOR DE LA BASE DE DATOS
- ABRIR LA BASE DE DATOS DE LA QUE QUEREMOS SACAR INFORMACIÓN
- EJECUTAR LAS INSTRUCCIONES **SQL** CONCRETAS PARA OBTENER LA INFORMACIÓN DESEADA
- DETENER EL MOTOR DE LA BASE DE DATOS

2.2 ENTORNO GRÁFICO DEL SGBD

CASI TODOS LOS SISTEMAS GESTORES DE BASES DE DATOS INCLUYEN ALGÚN ENTORNO GRÁFICO DESDE EL CUAL SE PUEDA ACCEDER DE FORMA DIRECTA Y MÁS CÓMODA A LA BASE DE DATOS

PASOS A SEGUIR:

- ESTABLECER LA CONEXIÓN A LA BASE DE DATOS DESEADA
- UNA VEZ LA CONEXIÓN SE HA ESTABLECIDO SOLO NOS QUEDA UTILIZAR LAS OPCIONES DE MENÚ Y HERRAMIENTAS GRÁFICAS DEL ENTORNO



The screenshot shows the phpMyAdmin interface for the 'information_schema' database. The left sidebar lists various tables under the 'information_schema' schema. The main area displays the 'CHARACTER_SETS' table in a grid format. The table has columns for 'Action', 'Lignes', 'Type', and 'Interclassement'. The data for 'CHARACTER_SETS' includes:

Action	Lignes	Type	Interclassement
	36	MEMORY	utf8_general_ci
	126	MEMORY	utf8_general_ci
	126	MEMORY	utf8_general_ci
	1 442	MyISAM	utf8_general_ci
	0	MEMORY	utf8_general_ci
	199	MEMORY	utf8_general_ci
	0	MyISAM	utf8_general_ci
	5	MEMORY	utf8_general_ci
	124	MEMORY	utf8_general_ci
	354	MEMORY	utf8_general_ci
	153	MEMORY	utf8_general_ci
	140	MEMORY	utf8_general_ci
	0	MEMORY	utf8_general_ci
	0	MyISAM	utf8_general_ci
	31	MEMORY	utf8_general_ci
	2	MyISAM	utf8_general_ci

At the bottom of the table, it says '16 table(s)' and 'Somme'. Below the table, there are links for 'Version imprimable' and 'Dictionnaire de données'.

2.3 DESDE PROGRAMAS SOFTWARE

CUALQUIER APLICACIÓN SOFTWARE UTILIZA DATOS POR LO QUE LOS PROGRAMAS QUE LA CONFORMAN DEBEN DISPONER DE INSTRUCCIONES PARA PODER REALIZAR ESE ACCESO

3. BASES DE DATOS EMBEBIDAS

- UNA BASE DE DATOS EMBEBIDA ES UNA BASE DE DATOS CUYO MOTOR ESTÁ **INCRUSTADO** EN LA APLICACIÓN, PERMITE TRABAJAR CON UNA BASE DE DATOS INSTALADA DIRECTAMENTE EN EL CLIENTE JUNTO CON LA APLICACIÓN QUE LA UTILIZA
- EL MOTOR DE LA BD SE INICIA CUANDO LA APLICACIÓN ARRANCA Y SE DETIENE CUANDO SE CIERRA LA APLICACIÓN

3.1 SQLITE



- ES UN **SGBD RELACIONAL** MULTIPLATAFORMA ESCRITO EN C QUE PROPORCIONA UN MOTOR MUY LIGERO PERO A LA VEZ POTENTE
- LAS BASES DE DATOS SE GUARDAN EN FORMA DE FICHEROS POR LO QUE ES FÁCIL TRASLADAR LA BASE DE DATOS CON LA APLICACIÓN QUE LA USA

[SITIO WEB DE SQLITE](#)

[DESCARGA](#)

3.1 SQLITE



- ADEMÁS DE COMO BASE DE DATOS EMBEBIDA, SQLITE NOS PERMITE UTILIZARLA COMO BASE DE DATOS EN MEMORIA

[SITIO WEB DE SQLITE](#)

[DESCARGA](#)

3.2 OTROS SGBD EMBEBIDOS

RELACIONALES

- [APACHE DERBY](#)
- [HSQLDB](#)
- [H2](#)
- [FIREBIRD EMBEDDED](#)

NO RELACIONALES

- [LMDB](#)
- [OBJECTDB](#)

HyperSQL



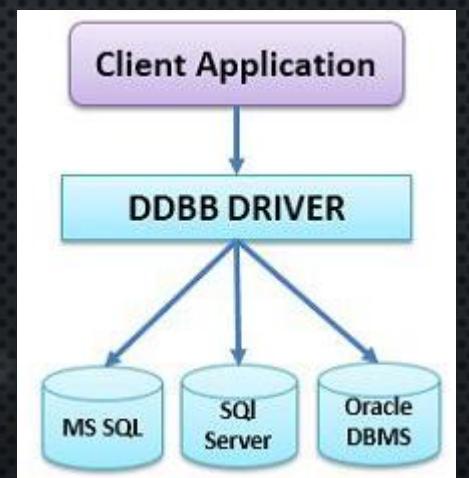
4. PROTOCOLOS DE ACCESO A DATOS

PERSPECTIVA HISTÓRICA:

1. CADA SGBD UTILIZABA PROTOCOLOS DE ACCESO ESPECÍFICOS DE LA MARCA QUE LOS FABRICABA
 - ERA NECESARIO APRENDER NUEVOS LENGUAJES SI SE MODIFICABA EL SGBD

SOLUCIÓN:

2. SE DEFINEN PROTOCOLOS DE ACCESO PARA PROPORCIONAR UNA INTERFAZ COMÚN
 - UN PROTOCOLO DE ACCESO A DATOS ES UNA CAPA INTERMEDIA ENTRE EL PROGRAMA DE USUARIO Y EL SGBD



4. PROTOCOLOS DE ACCESO A DATOS

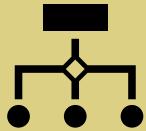
- **ODBC**
 - OPEN DATABASE CONNECTIVITY. API QUE PUEDEN USAR LAS APLICACIONES PARA MANEJAR UNA BD Y HACER CONSULTAS.
 - DEFINIDO POR MICROSOFT
 - MUY UTILIZADO EN WINDOWS
- **PERL:DBI**
 - PERL ES UN LENGUAJE PARA PROGRAMACIÓN WEB
 - MUY EXTENDIDO EN LINUX
- **JDBC**
 - JAVA DATABASE CONNECTIVITY, API PARA CLIENTES PROGRAMADOS EN JAVA PARA CONECTARSE A UNA BASE DE DATOS RELACIONALES.

ODBC

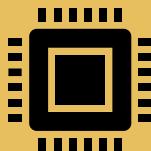
- ES LA SOLUCIÓN DE **MICROSOFT**
- SE APOYA EN UN AMPLIO CONJUNTO DE **DRIVERS ODBC**, COMO BIBLIOTECAS QUE SE UTILIZAN DESDE EL PROGRAMA CLIENTE.
- CADA DRIVER ESTÁ ORIENTADO A UN SISTEMA DE GESTIÓN DE BASE DE DATOS CONCRETO
- ODBC PERMITE LA CONEXIÓN FÁCIL DESDE VARIOS LENGUAJES DE PROGRAMACIÓN, Y TIENE MUCHA ACEPTACIÓN, SOBRE TODO EN ENTORNOS **WINDOWS**.
- VENTAJAS:
 - PERMITE EL PODER GESTIONAR UN AMPLIO RANGO DE DATOS CON UNA SOLA INTERFAZ.
 - AMPLIAMENTE EXTENDIDO, CON LO QUE MUCHAS SON LAS APLICACIONES QUE LO INCLUYEN COMO DRIVER DE ACCESO.
- DESVENTAJA:
 - LA CONSULTA PASA A TRAVÉS DE VARIAS CAPAS, LO QUE REDUCE LA EFICIENCIA.
 - SE LIMITA A DATOS RELACIONALES BASADOS EN LA SINTAXIS DE SQL.
 - LA API ODBC USA UNA INTERFAZ ESCRITA EN EL LENGUAJE **C** Y NO ES APROPIADA PARA SU USO DIRECTO DESDE JAVA POR PROBLEMAS DE SEGURIDAD, DIFICULTAD DE LA IMPLEMENTACIÓN, ROBUSTEZ EN LOS PROGRAMAS Y PORTABILIDAD DE LAS APLICACIONES.
 - ODBC ES UNA BIBLIOTECA DIFÍCIL DE UTILIZAR Y APRENDER.



Consta de un conjunto de clases e interfaces Java que nos permiten acceder de una forma genérica a las Bases de Datos independientemente del proveedor del SGBD

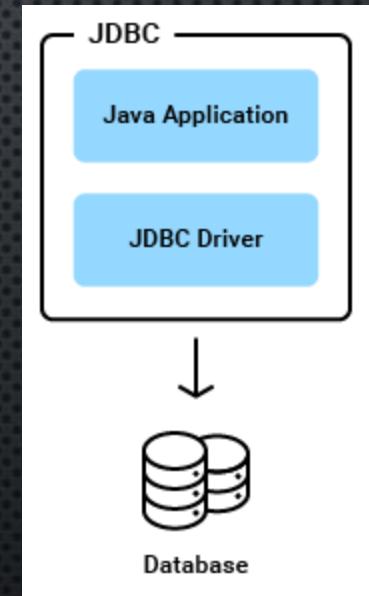


Cada proveedor del SGBD, dispondrá de una implementación de dichos interfaces. Dicha implementación se sabe comunicar con el SGBD de ese proveedor



Estas clases e interfaces JDBC se encuentran en el paquete `java.sql.*`

5. JDBC



5. JDBC



PASOS A SEGUIR:



Establecer una conexión con una Base de Datos



Crear y enviar una sentencia SQL a la Base de Datos



Procesar el resultado



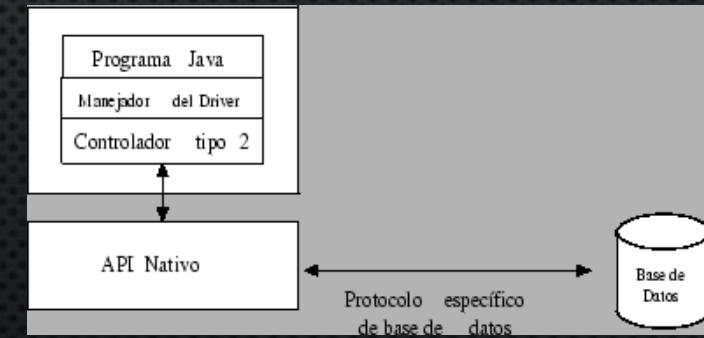
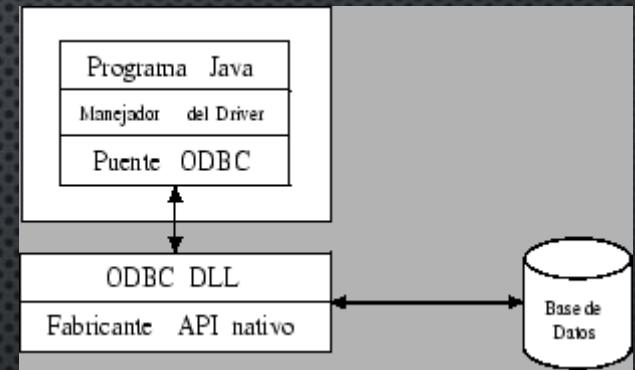
Cerrar la conexión

5.1 TIPOS DE DRIVERS JDBC

LOS **DRIVERS JDBC** SON LA IMPLEMENTACIÓN QUE CADA PROVEEDOR DEL SGBD HA REALIZADO DEL API JDBC

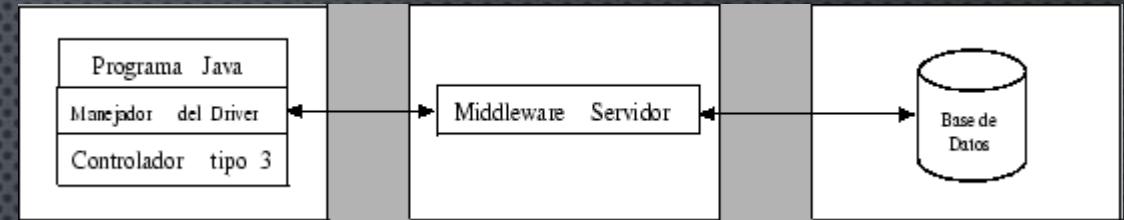
EXISTEN **CUATRO TIPOS**:

- **TIPO 1: JDBC-ODBC BRIDGE:** : TRADUCE OPERACIONES JDBC EN LLAMADAS A LA API ODBC. ESTAS LLAMADAS SON ENTONCES CURSADAS A LA BASE DE DATOS MEDIANTE EL DRIVER ODBC APROPIADO. NECESA INSTALACIÓN Y CONFIGURACIÓN DE ODBC EN LA MÁQUINA CLIENTE
- **TIPO 2: NATIVE-API PARTLY-JAVA:** : UTILIZA LA INTERFAZ DE MÉTODOS NATIVOS DE JAVA PARA CONVERTIR LAS SOLICITUDES DE API JDBC EN LLAMADAS ESPECÍFICAS Y NATIVAS DEL SGBD. EXIGE INSTALAR EN LA MÁQUINA CLIENTE CÓDIGO BINARIO (API) PROPIO DEL CLIENTE DE BASE DE DATOS Y PROPIO DEL SISTEMA OPERATIVO

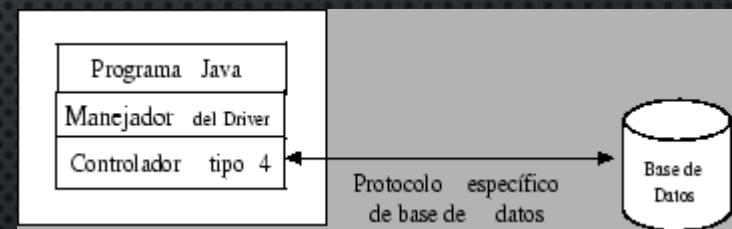


5.1 TIPOS DE DRIVERS JDBC

- TIPO 3: **JDBC-NET PURE JAVA**: IMPLEMENTADOS EN UNA APROXIMACIÓN DE TRES CAPAS POR LO QUE LAS SOLICITUDES DE LA BASE DE DATOS JDBC ESTÁN TRADUCIDAS EN UN PROTOCOLO DE RED INDEPENDIENTE DE LA BASE DE DATOS Y DIRIGIDAS AL SERVIDOR DE CAPA INTERMEDIA. EL SERVIDOR DE LA CAPA INTERMEDIA RECIBE LAS SOLICITUDES Y LAS ENVÍA A LA BASE DE DATOS UTILIZANDO PARA ELLO UN DRIVER JDBC DEL TIPO 1 O DEL TIPO 2 (LO QUE SIGNIFICA QUE SE TRATA DE UNA ARQUITECTURA MUY FLEXIBLE)



- TIPO 4: **NATIVE PROTOCOL PURE JAVA**: COMUNICA DIRECTAMENTE CON EL SERVIDOR DE BASES DE DATOS UTILIZANDO EL PROTOCOLO NATIVO DEL SERVIDOR. ESTOS DRIVERS PUEDEN ESCRIBIRSE TOTALMENTE EN JAVA, SON INDEPENDIENTES DE LA PLATAFORMA Y ELIMINAN TODO LOS ASPECTOS RELACIONADOS CON LA CONFIGURACIÓN EN EL CLIENTE. NO EXIGE INSTALACIÓN EN CLIENTE



Los tipos 3 y 4 son la mejor forma de acceder a bases de datos JDBC

Los SGBDs tendrán un fichero JAR o ZIP con las clases del Driver JDBC que habrá que añadir a la variable **CLASSPATH** del sistema para poder utilizarlo desde nuestras aplicaciones Java o almacenar una copia del JAR o ZIP en la carpeta **LIB** del proyecto Java

5.2 COMPONENTES JDBC

- EL GESTOR DE LOS DRIVERS (JAVA.SQL.DRIVERMANAGER).
- LA CONEXIÓN CON LA BASE DE DATOS (JAVA.SQL.CONNECTION).
- LA SENTENCIA A EJECUTAR (JAVA.SQL.STATEMENT).
- EL RESULTADO (JAVA.SQL.RESULTSET).

6. ESTABLECER CONEXIÓN A LA BASE DE DATOS

JAVA.SQL.DRIVERMANAGER

- El **DRIVER MANAGER** LLEVA EL CONTROL DE LOS DRIVERS JDBC CARGADOS EN MEMORIA. ADEMÁS ES EL ENCARGADO DE REALIZAR LAS CONEXIONES CON LA BASE DE DATOS.
- Es necesario cargar en memoria los drivers JDBC para registrarlos al **DRIVER MANAGER**.
- Se cargan mediante el método estático **FORNAME()** de la clase **java.lang.Class**, por ejemplo:

```
CLASS.forName("COM.MYSQL.JDBC.DRIVER");
```

CARGA EL DRIVER JDBC DEL SGBD MYSQL

Ya no es necesario!!
Se considera código
LEGACY

6. ESTABLECER CONEXIÓN A LA BASE DE DATOS

JAVA.SQL.CONNECTION

- REPRESENTA UNA CONEXIÓN CON LA BASE DE DATOS. EL ENCARGADO DE ABRIR UNA CONEXIÓN ES EL DRIVER MANAGER MEDIANTE EL MÉTODO ESTÁTICO:

```
public static Connection GetConnection(String url, String user, String pwd) throws SQLException
```

- URL: ES EL IDENTIFICADOR DE LA BASE DE DATOS.
- USER: USUARIO CON EL QUE SE ABRE LA CONEXIÓN (OPCIONAL)
- PWD: CONTRASEÑA DEL USUARIO (OPCIONAL)

ALTERNATIVAS

```
public static Connection getConnection(String url,  
java.util.Properties info) throws SQLException {
```

```
public static Connection getConnection(String url)  
throws SQLException {
```

6.1 CIERRE DE CONEXIÓN

- UNA VEZ HAYAMOS TERMINADO DE TRABAJAR CON UNA CONEXIÓN DEBEMOS LIBERARLA
- UNA CONEXIÓN ABIERTA SIGNIFICA RECURSOS CONSUMIÉNDOSE EN EL SGBD. LAS CONEXIONES SE CIERRAN MEDIANTE UNA LLAMADA AL MÉTODO:

```
public void close() throws SQLException
```

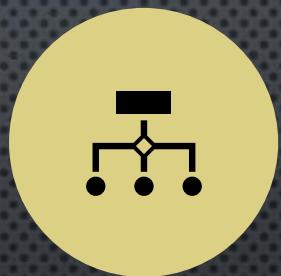
EJEMPLO

```
import java.sql.*;
public class EstablecerConexion {
    Run | Debug
    public static void main(String[] args)
    {
        try {
            Class.forName("com.ibm.db2.jcc.DB2Driver");
            Connection con = DriverManager.getConnection("jdbc:db2:sample");
            // ... aquí trataríamos los datos .....
            con.close();
        }
        catch(ClassNotFoundException ex)
        {
            ex.printStackTrace();
        }
        catch(SQLException ex)
        {
            ex.printStackTrace();
        }
    }
}
```

7. ACCESO A METADATOS



¿Qué son los metadatos?



Consisten en información sobre los datos. Incluye la estructura de los datos, relaciones y campos que forman las tablas



¿Qué utilidad tienen?



Debemos acceder a una base de datos, no tenemos un cliente de base de datos que nos permita averiguar la estructura => Mediante los metadatos podremos averiguarla

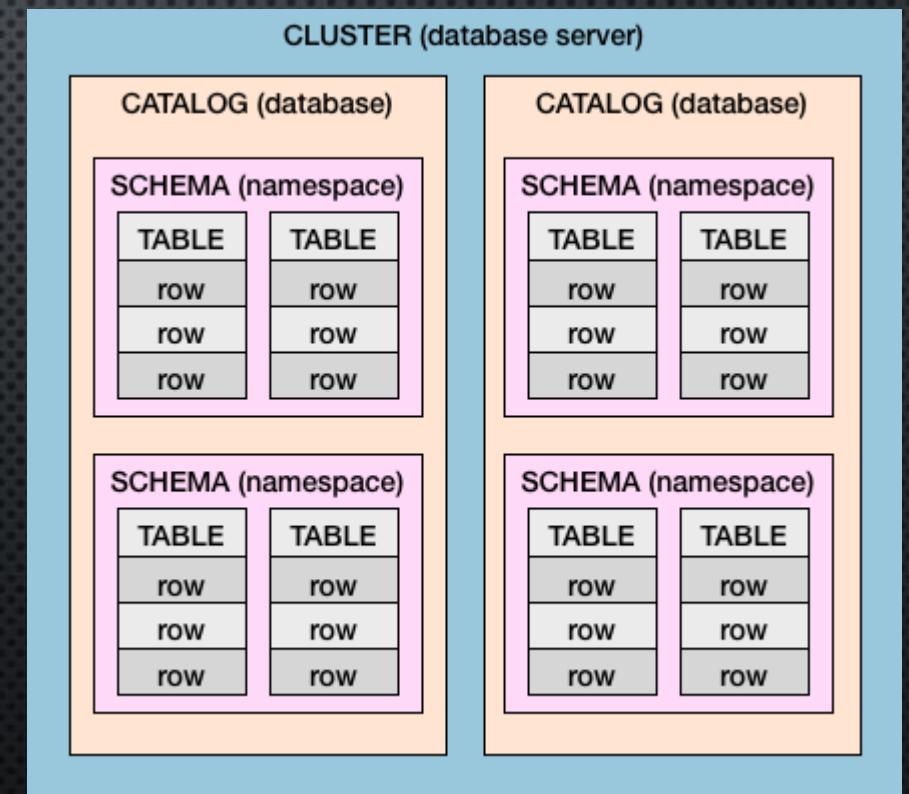
DatabaseMetaData

- PROPORCIONA UNA GRAN CANTIDAD DE INFORMACIÓN A TRAVÉS DE MÚLTIPLES MÉTODOS.
- ESTE OBJETO SE RECUPERA A TRAVÉS DEL MÉTODO **GETMETADATA()** DE LA CLASE **CONNECTION**.
- LOS NOMBRES DE LOS MÉTODOS QUE INCLUYE SON BASTANTE EXPLÍCITOS:
- `.GETDATABASEPRODUCTNAME()`, `.GETDRIVERNAME()`, `.GETURL()`, `.GETUSERNAME()`...
- [REFERENCIA API 11](#)
- [REFERENCIA API 8](#)

getTables()

DEVUELVE UN OBJETO RESULTSET QUE PROPORCIONA INFORMACIÓN SOBRE LAS TABLAS Y LAS VISTAS DE LA BASE DE DATOS

REFERENCIA API [1]



Ejemplo

SE LE PASA:

- CATALOG: LA BBDD. CON NULL INDICAMOS EL ACTUAL.
- SCHEMAPATTERN: ESQUEMA DEL BBDD.
- TABLENAMEPATTERN: PATRÓN PARA LA SELECCIÓN DE LA TABLA (CON NULL TODAS LAS TABLAS)
- TYPES: TIPOS DE OBJETOS A OBTENER (TABLE, VIEW, NULL TODOS)

Y SE PUEDE OBTENER:

- CATALOGO (LA BBDD).
- ESQUEMA (ESPACIO DE NOMBRES DE TABLAS)
- NOMBRE DE LA TABLA.
- TIPO DE TABLA.
- ...ENTRE OTROS.

```
import java.sql.*;
public class AccessMetadata {
    Run | Debug
    public static void main(String[] args)
    {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost/test?" +
                "user=minty&password=greatsqldb");
            DatabaseMetaData dbmd= con.getMetaData();
            ResultSet resul= dbmd.getTables(null, "mibd",null, null);
            while (resul.next())
            {
                String catalogo = resul.getString("TABLE_CAT");
                String esquema = resul.getString("TABLE_SCHEMA");
                String tabla = resul.getString("TABLE_NAME");
                String tipo = resul.getString("TABLE_TYPE");
                System.out.println("Catalogo: " + catalogo + " Esquema: " + esquema + " tipo: " +
                    tipo + " Nombre tabla: " + tabla);
            }
            con.close();
        }
        catch(ClassNotFoundException ex)
        {
            ex.printStackTrace();
        }
        catch(SQLException ex)
        {
            ex.printStackTrace();
        }
    }
}
```

getColumns()

`.GETCOLUMNS(CATALOGO, ESQUEMA, NOMBRETABLA, NOMBRECOLUMN)`

DEVUELVE UN OBJETO RESULTSET CON INFORMACIÓN SOBRE LAS COLUMNAS DE UNA TABLA O TABLAS. PARA EL NOMBRE DE LA TABLA Y COLUMNA SE PUEDEN UTILIZAR LOS COMODINES _ Y % (_:UN CARÁCTER, %: 0..N CARACTERES) Y PARA CUALQUIERA DE LOS PARÁMETROS SE PUEDE UTILIZAR EL VALOR NULL.

LOS DIFERENTES METADATOS A LOS QUE SE PUEDEN ACCEDER DE UNA COLUMNA SE PUEDEN VER EN:

- [REFERENCIA API 11](#)

Ejemplo

```
import java.sql.*;
public class AccessMetadataColumns {
    Run | Debug
    public static void main(String[] args)
    {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost/test?" +
                "user=minty&password=greatsqldb");
            DatabaseMetaData dbmd= con.getMetaData();
            ResultSet columnas = dbmd.getColumns(null, "mibd", "mitabla", null);
            System.out.println(" COLUMNAS TABLA mitabla ");
            while (columnas.next())
            {
                String nombreCol = columnas.getString("COLUMN_NAME");
                String tipoCol = columnas.getString("TYPE_NAME");
                String tamCol = columnas.getString("COLUMN_SIZE");
                String nula = columnas.getString("IS_NULLABLE");
                System.out.println("Columna: " + nombreCol + " tipo: " + tipoCol + " tamaño: " +
                    tamCol + " Admite null: " + nula);
            }
            con.close();
        }
        catch(ClassNotFoundException ex)
        {
            ex.printStackTrace();
        }
        catch(SQLException ex)
        {
            ex.printStackTrace();
        }
    }
}
```

OTROS MÉTODOS INTERESANTES

- `.GETPRIMARYKEYS(CATALOGO,ESQUEMA,TABLA)`: OBTIENE LAS CLAVES PRIMARIAS DE LA TABLA.
- `.GETEXPORTEDKEYS(CATALOGO,ESQUEMA,TABLA)`: OBTIENE LAS CLAVES PRIMARIAS QUE SON AJENAS EN OTRAS TABLAS.
- `.GETIMPORTEDKEYS(CATALOGO,ESQUEMA,TABLA)`: OBTIENE LAS CLAVES AJENAS QUE SON PRIMARIAS EN OTRAS TABLAS.
- `.GETPROCEDURES(CATALOGO,ESQUEMA,PROCEDIMIENTO)`: OBTIENE LOS PROCEDIMIENTOS DEFINIDOS EN LA TABLA (PROCEDIMIENTOS, FUNCIONES, TRIGGERS).
- TODOS ESTOS MÉTODOS DEVUELVEN UN `RESULTSET` EN LOS QUE SE PUEDE VER DIFERENTE INFORMACIÓN POR EJEMPLO CON `GetString("...")`; SE RECOMENDÁ CONSULTAR LOS DIFERENTES DATOS QUE DEVUELVEN EN EL SIGUIENTE ENLACE.
- [REFERENCIA API 11](#)

PRÁCTICA

- REALIZA LA PRÁCTICA 1.
- REALIZA LA PRÁCTICA 2.

8. EJECUCIÓN DE SENTENCIAS

- DEPENDIENDO DEL TIPO DE SENTENCIA A EJECUTAR UTILIZAREMOS UN INTERFACE U OTRO Y UTILIZAREMOS UN MÉTODO U OTRO PARA EJECUTARLA



8. EJECUCIÓN DE SENTENCIAS

PARA CREAR UN OBJETO BASADO EN UNA DE ESTAS INTERFACES USAREMOS EL MÉTODO CORRESPONDIENTE DEL OBJETO **CONNECTION**.

- **STATEMENT** SE USA PARA EJECUTAR UNA SENTENCIA SQL SIN INFORMACIÓN DINÁMICA (PARÁMETROS) CONTRA LA BD
- **PREPAREDSTATEMENT** HEREDA DE **JAVA.SQL.STATEMENT**. SE DIFERENCIA DE ELLA EN QUE:
 - LA SENTENCIA SQL ES COMPILADA POR EL SGBD PREVIAMENTE
 - LA SENTENCIA PUEDE CONTENER VARIABLES MARCADAS COMO PARÁMETROS
- **CALLABLESTATEMENT** HEREDA DE **JAVA.SQL.PREPAREDSTATEMENT**, Y SIRVE PARA EJECUTAR PROCEDIMIENTOS ALMACENADOS EN LA BD

8.1 ResultSet

- REPRESENTA UN CONJUNTO DE FILAS CON SUS COLUMNAS, EN EL CASO DE UTILIZARLOS PARA ALMACENAR EL RESULTADO DE LA EJECUCIÓN DE UN STATEMENT CON LA INSTRUCCIÓN SELECT CONTENDRÁ TODAS LAS FILAS RESULTANTES DE LA SELECT
- IMPLEMENTA MÉTODOS PARA:
 - ACCEDER A LAS FILAS QUE COMPONEN EL RESULTADO
 - ACCEDER AL VALOR DE CADA COLUMNA DE LA FILA SELECCIONADA

8.1 ResultSet

EL MÉTODO PARA ACCEDER A LA FILA SIGUIENTE DEL RESULTADO ES:

```
public boolean next() throws SQLException;
```

EL MÉTODO PARA ACCEDER A LA FILA ANTERIOR DEL RESULTADO ES:

```
public boolean previous() throws SQLException;
```

EL MÉTODO A USAR PARA ACCEDER AL VALOR DE UNA COLUMNA, DEPENDERÁ DEL TIPO DE DATO DE LA COLUMNA (xxxx), SIENDO 1 EL VALOR DEL ÍNDICE PARA LA PRIMERA COLUMNA.

```
public xxxx getxxxx(int column) throws SQLException;
```

8.1 ResultSet

El ResultSet ACTUA COMO UN CURSOR Y CUANDO SE CREA EL CURSOR ESTÁ POSICIONADO ANTES DE LA PRIMERA FILA, POR LO QUE HABRÁ QUE EJECUTAR UN PRIMER NEXT() PARA ACCEDER A LA PRIMERA FILA.

SE SUELE IMPLEMENTAR UN BUCLE DE TIPO WHILE PARA RECORRER Y TRATAR TODO EL ResultSet.

Los ResultSets SE CIERRAN MEDIANTE EL MÉTODO:

```
public boolean close() throws SQLException;
```

NOTA: EL ResultSet SE CIERRA AUTOMÁTICAMENTE AL CERRAR EL STATEMENT QUE LO CREÓ. NO OBSTANTE, ES UNA BUENA PRÁCTICA CERRARLO MANUALMENTE

8.1 ResultSet

EL ResultSet ACTUA COMO UN CURSOR Y CUANDO SE CREA EL CURSOR ESTÁ POSICIONADO ANTES DE LA PRIMERA FILA, POR LO QUE HABRÁ QUE EJECUTAR UN PRIMER NEXT() PARA ACCEDER A LA PRIMERA FILA.

SE SUELE IMPLEMENTAR UN BUCLE DE TIPO WHILE PARA RECORRER Y TRATAR TODO EL ResultSet.

Los ResultSets SE CIERRAN MEDIANTE EL MÉTODO:

```
public boolean close() throws SQLException;
```

NOTA: EL ResultSet SE CIERRA AUTOMÁTICAMENTE AL CERRAR EL STATEMENT QUE LO CREÓ. NO OBSTANTE, ES UNA BUENA PRÁCTICA CERRARLO MANUALMENTE

Ejemplo

ResultSet

```
import java.sql.*;
public class EjemploResultSet {
    public static void main (String[] args) throws ClassNotFoundException, SQLException {
        //CONEXIÓN A MYSQL
        Class.forName("com.mysql.jdbc.Driver");
        Connection conexión = DriverManager.getConnection ("jdbc:mysql://localhost/ejemplo","ejemplo", "ejemplo");
        String sql="SELECT * FROM mitable";
        Statement sentencia = conexión.createStatement();
        ResultSet rs = sentencia.executeQuery(sql);

        while (rs.next())
            System.out.printf("%d, %s, %s %n", rs.getInt(1), rs.getString(2), rs.getString(3)); rs.close();

        sentencia.close();
        conexión.close();
    } //main
}
```

8.2 Statement

- SE UTILIZA PARA EJECUTAR UNA SENTENCIA SQL SIN PARÁMETROS
- SIEMPRE LLEVA ASOCIADA UNA CONEXIÓN QUE SIRVIÓ PARA CREARLO
- LOS OBJETOS BASADOS EN ESTA INTERFACE SE CREAN CON EL SIGUIENTE MÉTODO DE LA CLASE JAVA.SQL.CONNECTION:

```
public Statement createStatement() throws SQLException;
```

EJEMPLO:

```
Statement sentencia = miconexion.createStatement();
```

SE CIERRAN MEDIANTE EL MÉTODO:

```
public void close() throws java.sql.SQLException;
```

8.2 Statement

EL MÉTODO PARA EJECUTARLO, DEPENDE DEL TIPO DE SENTENCIA SQL QUE CONTENGA
SENTENCIA SELECT:

- SE USA EL MÉTODO: `EXECUTEQUERY(STRING SQL)`;
- DEVUELVE UNA INSTANCIA DE `JAVA.SQL.RESULTSET`

SENTENCIA DML(INSERT, UPDATE Y DELETE) Y DDL (CREATE, DROP, ALTER):

- SE USA EL MÉTODO: `EXECUTEUPDATE(STRING SQL)`;
- DEVUELVE UN `INT` CON EL NÚMERO DE FILAS AFECTADAS

SENTENCIA GENERICA:

- SE USA EL MÉTODO: `EXECUTE(STRING SQL)`;
- DEVUELVE:
 - TRUE SI DEVUELVE UN `RESULTSET`, QUE HABRÁ QUE RECUPERAR CON `GETRESULTSET()`.
 - FALSE SI DEVUELVE EL NÚMERO DE FILAS AFECTADAS QUE HABRÁ QUE RECUPERAR CON `GETUPDATE()`, O NO HAY RESULTADOS.

PRÁCTICA

- REALIZA LA PRÁCTICA 3.

8.3 PreparedStatement

- SE DIFERENCIA DE JAVA.SQL.**STATEMENT** EN DOS COSAS:
 - LA SENTENCIA SQL ES COMPILADA POR EL SGBD PREVIAMENTE.
 - LA SENTENCIA PUEDE CONTENER PARÁMETROS
- SIEMPRE LLEVA ASOCIADA UNA CONEXIÓN QUE SIRVIÓ COMO ORIGEN PARA SU CREACIÓN
- SE CREAN CON EL SIGUIENTE MÉTODO DE LA CLASE JAVA.SQL.**CONNECTION**:

```
public PreparedStatement prepareStatement(String sql) throws SQLException;
```

- A ESTE MÉTODO SE LE PASA UN STRING DE LA FORMA:
 - **String sql = “INSERT INTO tabla VALUES (?, ?, ?)”**
 - “?” NO PUEDE REPRESENTAR UNA TABLA O COLUMNA, POR EJEMPLO “SELECT * FROM ?” Es incorrecto.

8.3 PreparedStatement

PARA EJECUTAR LA SENTENCIA SE PUEDEN UTILIZAR LOS MISMOS MÉTODOS QUE PARA STATEMENT PERO SIN CADENA SQL YA QUE LA SQL LA HABREMOS RELLENADO CON EL MÉTODO PREPARESTATEMENT():

```
public ResultSet executeQuery() throws SQLException;  
public ResultSet executeUpdate() throws SQLException;  
public boolean execute() throws SQLException;
```

EN EL CASO DE QUE LA SENTENCIA CONTENGA ALGÚN PARÁMETRO, EL VALOR DE DICHO PARÁMETRO SE TENDRÁ QUE ASIGNAR ANTES DE EJECUTAR LA SENTENCIA CON EL MÉTODO DE LA CLASE JAVA.SQL.[PREPAREDSTATEMENT](#):

```
public void setXXXX(int col, xxxx value) throws SQLException;
```

EL NÚMERO DE COLUMNA EMPIEZA EN 1.

PRÁCTICA

- REALIZA LA PRÁCTICA 4.

8.4 CallableStatement

- HEREDA DE `JAVA.SQL.PREPAREDSTATEMENT`, Y SIRVE PARA EJECUTAR PROCEDIMIENTOS ALMACENADOS EN LA BASE DE DATOS
- UN PROCEDIMIENTO ALMACENADO ES UN PROGRAMA QUE RESIDE Y SE EJECUTA EN EL PROPIO SGBD. SE PUEDE DEFINIR CON PARÁMETROS DE ENTRADA, CON PARÁMETROS DE SALIDA, O AMBOS, O SIN NINGUNO
- TAMBIÉN PUEDEN DEVOLVER UN VALOR, EN ESTE CASO ACTÚA COMO UNA FUNCIÓN

8.4 CallableStatement

COMO SON DEFINIDOS EN EL PROPIO SGBD, LOS TIPOS DE PARÁMETROS QUE ADMITAN
DEPENDERÁN DE LO QUE ADMITA EL SGBD

SE CREAN CON EL SIGUIENTE MÉTODO DE LA CLASE `JAVA.SQL.CONNECTION`:

```
public CallableStatement prepareCall(String call) throws SQLException;
```

8.4 CallableStatement

EL PARÁMETRO `String call` PASADO A LA LLAMADA `prepareCall` PUEDEN TENER TRES FORMAS:

- PROCEDIMIENTO CON PARÁMETROS DE ENTRADA Y/O SALIDA:

```
{call nombre_procedimiento(?, ?, ...)}
```

- FUNCIÓN CON PARÁMETROS DE SALIDA:

```
{? = call nombre_funcion}
```

- FUNCIÓN CON PARÁMETROS DE ENTRADA Y SALIDA:

```
{? = call nombre_funcion(?, ?, ...)}
```

- PROCEDIMIENTO SIN PARÁMETROS:

```
{call nombre_procedimiento}
```

8.4 CallableStatement

PARA EJECUTAR LA SENTENCIA SE LLAMA AL MÉTODO

```
public boolean execute() throws SQLException;
```

ANTES DE EJECUTAR LA SENTENCIA NOS TENEMOS QUE ASEGURAR DE HABER RELLENADO LOS PARÁMETROS CON LOS MÉTODOS CORRESPONDIENTES:

PARA PARÁMETROS DE ENTRADA: .[SETXXX](#)

```
public void setXXXX(int col, xxxx value) throws  
SQLException;
```

PARA PARÁMETROS DE SALIDA: .[REGISTEROUTPARAMETER](#)

```
RegisterOutParameter(int parameterIndex, int sqlType)  
sqlType es un enumerado (java.sql.sqlType)
```

8.4 CallableStatement

```
String sql = "{? = call numCharacters()}";
CallableStatement stm = conexion.prepareCall(sql);
stm.registerOutParameter(1, java.sql.Types.INTEGER);
stm.execute();
System.out.println(stm.getInt(1));
stm.close();
```

```
DELIMITER $$  
CREATE FUNCTION `numCharacters`() RETURNS int(11)  
BEGIN  
    RETURN (SELECT COUNT(*) FROM characters);  
END$$  
DELIMITER ;|
```

PARA PROCESAR EL RESULTADO DE LA EJECUCIÓN DE LA SENTENCIA, SOBRE EL PROPIO CALLABLESTATEMENT ACCEDEMOS A LOS DATOS POR ÍNDICE MEDIANTE MÉTODOS GETXXXX

ResultSetMetaData

- PODEMOS CONSEGUIR INFORMACIÓN MÁS DETALLADAS SOBRE LAS COLUMNAS DE UNA TABLA EN UN OBJETO **RESULTSETMETADATA** EJECUTANDO EL MÉTODO `.GETMETADATA()` DEL **RESULTSET**.
- CON UN OBJETO **RESULTSETMETADATA** PODEMOS RECUPERAR LAS PROPIEDADES DE LAS COLUMNAS DE UNA TABLA O TABLAS.

ResultSetMetaData

PASOS A SEGUIR:

- OBTENER EN UN RESULTSET LA COLUMNAS DE LA TABLA
- EJECUTAR SOBRE EL RESULTSET EL MÉTODO `.GETMETADATA()` GUARDANDO EL RESULTADO EN UN OBJETO `RESULTSETMETADATA`
- UTILIZAR LOS MÉTODOS DEL `RESULTSETMETADATA` PARA OBTENER LA INFORMACIÓN DESEADA

Ejemplo

ResultSetMetadata

```
import java.sql.*;
public class AccessResultSetMetaData {
    Run | Debug
    public static void main(String[] args)
    {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost/test?" +
                "user=minty&password=greatsqldb");
            Statement sentencia= con.createStatement();
            ResultSet resultado= sentencia.executeQuery("SELECT * FROM mitabla");
            ResultSetMetaData rsmd= resultado.getMetaData();
            int ncols= rsmd.getColumnCount();
            System.out.println("Núm. de columnas recuperadas: " + ncols);
            for (int i=1;i<=ncols;i++){
                System.out.println("Columna " + i + ":");
                System.out.println("Nombre: " + rsmd.getColumnName(i));
                System.out.println("Tipo: " + rsmd.getColumnTypeName(i));
            }
            con.close();
        }
        catch(ClassNotFoundException ex)
        {
            ex.printStackTrace();
        }
        catch(SQLException ex)
        {
            ex.printStackTrace();
        }
    }
}
```

9. GESTIÓN DE TRANSACCIONES

UNO DE LOS OBJETIVOS DE UN SGBD ES QUE MÚLTIPLES USUARIOS PUEDAN ACCEDER A LA BASE DE DATOS DE FORMA SIMULTÁNEA SIN QUE EL TRABAJO DE UNO INTERFIERA EN EL TRABAJO DEL OTRO

UN PROBLEMA QUE PUEDE APARECER CUANDO DOS USUARIOS ACCEDEN A LA MISMA BASE DE DATOS ES QUE QUIERAN ACCEDER A LOS MISMOS DATOS Y QUE AL MENOS UNO QUIERA ACTUALIZAR ESOS DATOS. SI ESTO NO SE CONTROLA DE FORMA IDÓNEA PODRÍAN APARECER PROBLEMAS DE INCONSISTENCIA DE LA BASE DE DATOS, POR ESO LOS SGBD SUELEN INCORPORAR SISTEMAS DE BLOQUEOS DE REGISTROS PARA LIMITAR ESTE TIPO DE ERRORES

CONTROL DE LA CONCURRENCIA

9. GESTIÓN DE TRANSACCIONES

UNA **TRANSACCIÓN** SE PUEDE DEFINIR COMO UN CONJUNTO DE ACCIONES QUE SE TIENEN QUE REALIZAR TODAS O NINGUNA PARA PRESERVAR LA **INTEGRIDAD** DE LA BASE DE DATOS

9.1 PROPIEDADES ACID

- **ATOMICIDAD.** UNA TRANSACCIÓN ES CONSIDERADA COMO UNA ÚNICA OPERACIÓN SOBRE LOS DATOS, SI SE COMPONE DE VARIAS OPERACIONES SE DEBERÁN EJECUTAR TODAS O NINGUNA
- **CONSISTENCIA.** DEBE PRESERVAR LA INTEGRIDAD DE LA BASE DE DATOS
- **AISLAMIENTO.** LAS TRANSACCIONES NO DEBEN INTERFERIR ENTRE SÍ
- **DURABILIDAD.** UNA VEZ QUE SE CONFIRMA UNA TRANSACCIÓN, SE GARANTIZA QUE SUS EFECTOS PERSISTAN INCLUSO EN EL CASO DE FALLOS POSTERIORES DEL SISTEMA

9.2 NIVELES DE AISLAMIENTO

- ASOCIADO A UNA TRANSACCIÓN ESTÁ SU NIVEL DE AISLAMIENTO QUE DEFINE EL GRADO EN QUE SE DEBE AISLAR LA TRANSACCIÓN DE LAS MODIFICACIONES DE RECURSOS O DATOS REALIZADAS POR OTRAS TRANSACCIONES.
- EL ACCESO SIMULTÁNEO A LA MISMA INFORMACIÓN PUEDE GENERAR EFECTOS SECUNDARIOS NO DESEABLES:
 - **LECTURAS DE DATOS SUCIOS**, LECTURA DE DATOS MODIFICADOS POR OTRA TRANSACCIÓN PERO NO CONFIRMADOS
 - **LECTURAS NO REPETIBLES**, “RELECTURA” DE FILAS QUE HAN SIDO MODIFICADAS O ELIMINADAS POR OTRA TRANSACCIÓN. FILAS QUE NO SE PUEDEN VOLVER A LEER
 - **LECTURAS FANTASMA**, LA TRANSACCIÓN VUELVE A EJECUTAR UNA CONSULTA QUE DEVUELVE UN CONJUNTO DE FILAS QUE RESPONDEN A UNA DETERMINADA CONDICIÓN Y ENCUENTRA QUE OTRA TRANSACCIÓN COMPLETADA HA INSERTADO NUEVAS FILAS QUE RESPONDEN TAMBIÉN A LA CONDICIÓN. FILAS QUE ANTES NO ESTABAN Y AHORA ESTÁN

9.2 NIVELES DE AISLAMIENTO

LOS NIVELES DE AISLAMIENTO DE LAS TRANSACCIONES CONTROLAN:

- SI SE REALIZAN BLOQUEOS CUANDO SE LEEN LOS DATOS Y QUÉ TIPOS DE BLOQUEOS SE SOLICITAN
- DURACIÓN DE LOS BLOQUEOS DE LECTURA
- SI UNA OPERACIÓN DE LECTURA QUE HACE REFERENCIA A FILAS MODIFICADAS POR OTRA TRANSACCIÓN:
 - SE BLOQUEA HASTA QUE SE LIBERA EL BLOQUEO EXCLUSIVO DE LA FILA.
 - RECUPERA LA VERSIÓN CONFIRMADA DE LA FILA QUE EXISTÍA EN EL MOMENTO EN EL QUE SE INICIÓ LA INSTRUCCIÓN O LA TRANSACCIÓN.
 - LEE LA MODIFICACIÓN DE LOS DATOS NO CONFIRMADA

9.2 NIVELES DE AISLAMIENTO

LOS NIVELES DE AISLAMIENTO SEGÚN EL ESTÁNDAR ANSI SQL-92 ESTÁN BASADOS EN QUÉ EFECTOS SECUNDARIOS PERMITEN:

Nivel aislamiento \	Permite Datos sucios	Lecturas no repetibles	Lecturas fantasma
Read Uncommitted	SI	SI	SI
Read Committed	NO	SI	SI
Repeatable Read	NO	NO	SI
Serializable	NO	NO	NO

9.3 OPERATIVA GENERAL CON TRANSACCIONES

PODEMOS ENCONTRAR TRANSACCIONES TANTO A NIVEL DE SQL COMO EN OTROS LENGUAJES PERO EN CUALQUIER CASO EL MODO DE OPERAR SERÁ EL MISMO:

- DETERMINAR EL **NIVEL DE AISLAMIENTO** DE LA TRANSACCIÓN
- INICIAR LA **TRANSACCIÓN**
- INCLUIR LAS OPERACIONES QUE VAN INCLUIDAS EN LA TRANSACCIÓN Y QUE SE TIENEN QUE EJECUTAR COMO UN TODO
- **CONFIRMAR** LA TRANSACCIÓN, FINALIZAR LA TRANSACCIÓN INDICANDO QUE LOS CAMBIOS QUEDEN PERMANENTES, SE SUELE INDICAR MEDIANTE LA ORDEN **COMMIT**
- O BIEN **ANULAR** LA TRANSACCIÓN, FINALIZAR LA TRANSACCIÓN INDICANDO QUE SE TIENE QUE DESHACER TODOS LOS CAMBIOS REALIZADOS HASTA AHORA DENTRO DE LA TRANSACCIÓN (SE REVIERTEN TODAS LAS MODIFICACIONES), SE SUELE INDICAR MEDIANTE LA ORDEN **ROLLBACK**
- SI DURANTE LA TRANSACCIÓN, EL PROCESO QUE LA LANZÓ SE INTERRUMPE, SE EFECTÚA UN **ROLLBACK IMPLÍCITO**

9.4 TRANSACCIONES EN JDBC

- AL CREAR UNA CONEXIÓN SE ESTÁ EN MODO AUTO-COMMIT QUE SIGNIFICA QUE CADA INSTRUCCIÓN SQL SE TRATA COMO UNA TRANSACCIÓN, AL LLEGAR AL PUNTO Y COMA, SE EJECUTA Y CONFIRMA LA TRANSACCIÓN.
- PARA PERMITIR AGRUPAR VARIAS INSTRUCCIONES EN UNA TRANSACCIÓN SE DEBE DESACTIVAR ESE MODO CON EL MÉTODO SETAUTOCOMMIT DE LA CONEXIÓN:

CONEXIÓN.SETAUTOCOMMIT(FALSE);

- PARA FINALIZAR LA TRANSACCIÓN UTILIZAMOS LOS MÉTODOS COMMIT (CONFIRMAR, DAR POR BUENOS LOS CAMBIOS) Y ROLLBACK (DESHACER LOS CAMBIOS) DE LA CONEXIÓN:

CONEXION.COMMIT; Y CONEXION.ROLLBACK;

- UTILIZAMOS CONEXIÓN.SETAUTOCOMMIT(TRUE); PARA VOLVER AL ESTADO POR DEFECTO QUE ES: UN COMMIT POR CADA INSTRUCCIÓN

9.4 TRANSACCIONES EN JDBC

- TENIENDO EN CUENTA QUE LAS **TRANSACCIONES** PUEDEN BLOQUEAR OTROS PROCESOS QUE USEN LOS MISMOS DATOS, SE DEBE **LIMITAR** EL USO DE TRANSACCIONES CON MÚLTIPLES INSTRUCCIONES A CUANDO SEA NECESARIO
- LOS **NIVELES DE AISLAMIENTO** PERMITIDOS DEPENDEN DEL SISTEMA GESTOR DE LA BASE DE DATOS. SE PUEDEN CONSULTAR MEDIANTE:

DATABASEMETADATA.SUPPORTSTRANSACTIONISOLATIONLEVEL

- EL NIVEL DE AISLAMIENTO UTILIZADO EN LA TRANSACCIÓN SE PUEDE OBTENER MEDIANTE EL MÉTODO **GETTRANSACTIONISOLATION** DE LA **CONEXIÓN** Y SE ESTABLECE MEDIANTE EL MÉTODO **SETTRANSACTIONISOLATION** TAMBIÉN DE LA **CONEXIÓN**

REFERENCIAS

- APUNTES DEL DEPARTAMENTO DE INFORMÁTICA Y COMUNICACIONES
- API DE JAVA (8 Y 11) [PACKAGE JAVA.SQL](#)
- [Docs ORACLE: JDBC BASICS](#)
- [Docs ORACLE: USING TRANSACTIONS](#)
- ACCESO A DATOS. EDITORIAL GARCETA
- [ACCESO A BASES DE DATOS CON JDBC. JUAN PAVÓN, UNIVERSIDAD COMPLUTENSE MADRID](#)
- [MYSQL CONNECTOR J](#)