

Unidad 14



Gestión de base de datos relacionales

Programación



Índice



- 14.1. Base de datos relacional
- 14.2. Creación de bases de datos
- 14.3. Establecimiento de la conexión
- 14.4. Ejecución de consultas sobre la base de datos
- 14.5. Mecanismos de actualización de la base de datos
- 14.6. Recuperación de la información
- 14.7. Manipulación de la información
- 14.8. Utilización de asistentes



Introducción

En la anterior unidad se introdujo las bases de datos orientadas a objetos, ahora veremos otro tipo de base de datos, las relacionales.

Desde el punto de vista de la programación trabajar con un tipo de base de datos u otra no dista de muchas diferencias. Hay que aprender a realizar la conexión a la base de datos, aplicar el sistema de gestión de la información, insertar, consultar, modificar y eliminar datos y, por último, realizar la desconexión de la base de datos.

En esta unidad se trabajará con conceptos básicos de base de datos, para tratar temas más específicos se pueden ver en la asignatura de base de datos.

Al finalizar esta unidad

- + Conoceremos las bases de datos relacionales
- + Instalaremos una base de datos.
- + Aprenderemos a realizar conexiones a las bases de datos mediante drivers
- + Aplicaremos el lenguaje SQL desde Java
- + Utilizaremos clientes de comandos y gráficos para base de datos



14.1.

Base de datos relacional

Las bases de datos relacionales son las más extendidas y usadas en el mercado, se diferencian de las orientadas a objetos en la forma como almacenan y dan acceso a los datos, que están relacionados entre sí, aplicando el modelo relacional.

Las principales características de una base de datos relacional:

- > La información se estructura en tablas y las relacionales que la componen.
- > Cada tabla esta es un conjunto de campos (columnas) y registros (filas).
- > Existen claves primarias o PK dentro de una tabla que identifican unequivocamente un registro.
- > Las relaciones se establecen a partir de las claves primarias.
- > Evitan duplicidad de registros de información.
- > Garantiza la integridad referencial.

Existen diferentes gestores de base de datos relacionales, Microsoft SQL Server, Oracle, DB2, PostgreSQL, MariaDB y MySQL. Para hacer esta unidad se ha basado en la base de datos MariaDB, de licencia libre.

14.2.

Creación de bases de datos

Previamente es necesario instalar el gestor de base de datos. La gran mayoría permiten desde su página web descargar un instalador con asistente, de esta manera se hace sencilla la tarea de instalarlo y configurarlo.

En la imagen nº 1 se muestra los pasos del asistente para su instalar MariaDB en Windows.

PARA AMPLIAR CONOCIMIENTO...

Para saber más de MariaDB dirigiros al siguiente enlace.
<https://mariadb.org/>

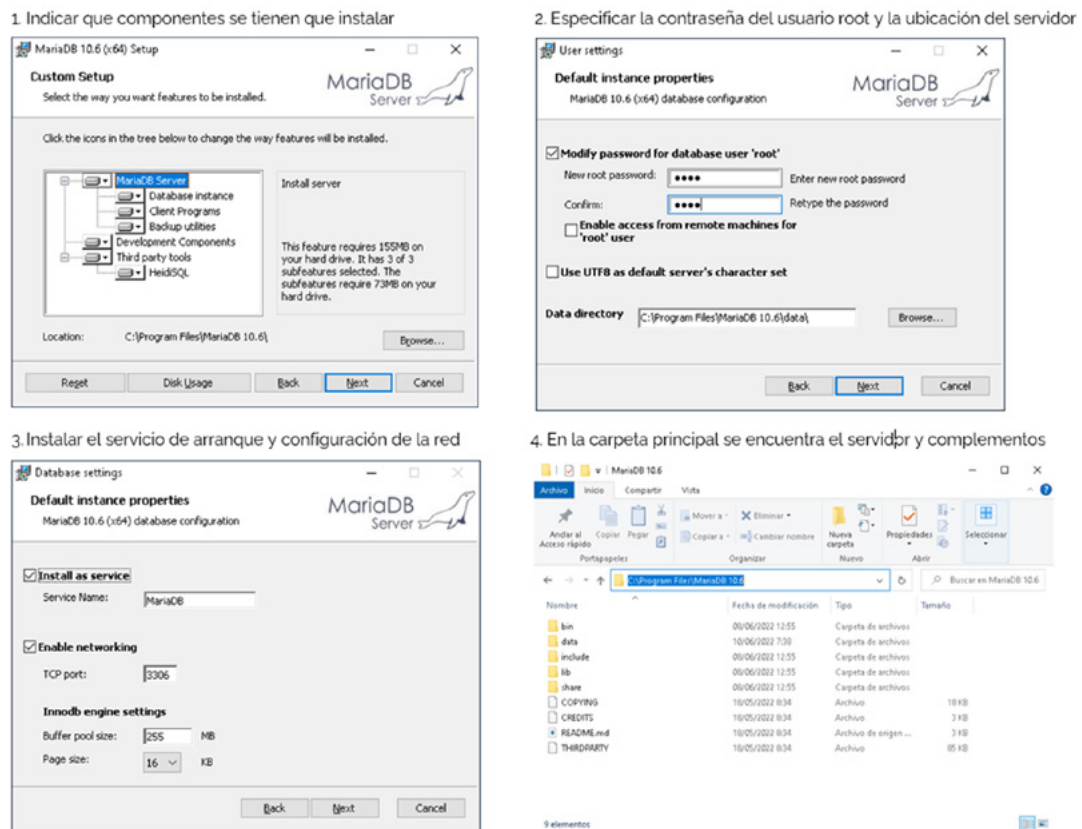


Imagen 1. Asistente de instalación de MariaDB

Una vez instalado el servidor de MariaDB se puede hacer uso de MySQL Client, un cliente en línea de comandos que nos permite trabajar directamente con el servidor y crear base de datos, tablas, etc.

En las imágenes nº 2 y 3 se muestra cómo crear una base de datos y las tablas que se usarán en esta unidad.

```
MySQL Client - "C:\Program Files\MariaDB 10.6\bin\mysql.exe" "--defaults-file=C:\Program Files\MariaDB 10.6\data\my.ini" -uroot -p
Enter password: ****
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.6.8-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE agenda;
Query OK, 1 row affected (0.038 sec)

MariaDB [(none)]> USE agenda;
Database changed
```

Imagen 2. Crear una base de datos agenda.

```
MySQL Client - "C:\Program Files\MariaDB 10.6\bin\mysql.exe" "--defaults-file=C:\Program File...
Your MariaDB connection id is 5
Server version: 10.6.8-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use agenda;
Database changed
MariaDB [agenda]> CREATE TABLE Contactos (
-> id INT AUTO_INCREMENT PRIMARY KEY,
-> nombre VARCHAR(255) NOT NULL,
-> apellido1 VARCHAR(255),
-> apellido2 VARCHAR(255),
-> telefono INT,
-> email VARCHAR(255)
-> );
Query OK, 0 rows affected (0.026 sec)

MariaDB [agenda]> CREATE TABLE tareas (
-> id INT AUTO_INCREMENT PRIMARY KEY,
-> nombre VARCHAR(255) NOT NULL,
-> descripcion VARCHAR(255)
-> );
Query OK, 0 rows affected (0.018 sec)

MariaDB [agenda]> CREATE TABLE historico (
-> id INT AUTO_INCREMENT PRIMARY KEY,
-> id_contacto INT NOT NULL,
-> id_tarea INT NOT NULL,
-> fecha DATE,
-> finalizada BOOLEAN NOT NULL DEFAULT FALSE,
-> FOREIGN KEY (id_contacto) REFERENCES contactos (id),
-> FOREIGN KEY (id_tarea) REFERENCES tareas (id)
-> );
Query OK, 0 rows affected (0.023 sec)

MariaDB [agenda]> _
```

Imagen 3. Crear las tablas contactos, tareas, histórico.

14.3.

Establecimiento de la conexión

Para establecer cualquier conexión a una base de datos desde Java, es necesario usar un driver específico dependiendo del tipo de base de datos y proveedor. Los drivers son librerías en ficheros .jar que están disponibles en las páginas web oficiales de cada proveedor.

Con la librería se cargará al proyecto de java desde el Java Build Path pestaña libraries.

Para MariaDB el driver es mariadb-java-client-(versionXX).jar .

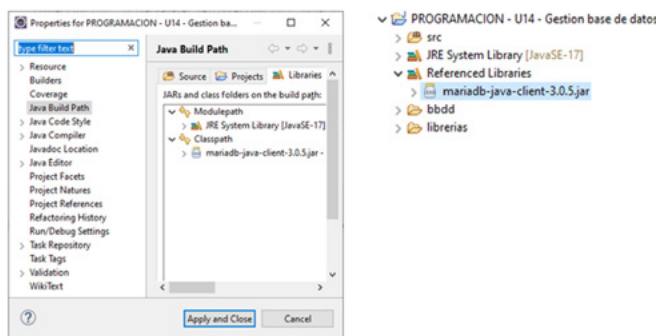


Imagen 4. Añadir la librería del driver MariaDB

Una vez cargada la librería con el driver específico a nuestro proyecto se establecerá la conexión desde java siguiendo el siguiente procedimiento:

1. **Registrar el driver**
2. **Montar la cadena de conexión.** Es un String con la url de la base de datos, en la que se especifica el protocolo (jdbc:"nombre motor base de datos"), la dirección ip, el puerto y el nombre de la base de datos o esquema.
3. **Crear la conexión** con DriverManager, usando la cadena de conexión y el usuario y contraseña que usa la base de datos.
4. **Crear la declaración** de trabajo (Statement) con la conexión para realizar las operaciones con las bases de datos
5. Finalmente **cerrar la declaración y la conexión** con el método close().

```
try {
    // 1.- Registrar el driver
    Class.forName("org.mariadb.jdbc.Driver");

    // 2.- Montar la cadena de conexión
    String url = "jdbc:mariadb://localhost:3306/agenda";
    String usuario = "usuario";
    String contraseña = "1234";

    // 3.- Crear la conexión
    Connection conexion = DriverManager.getConnection (url, usuario, contraseña);

    // 4.- Crear la declaración de trabajo
    Statement declaracion = conexion.createStatement();

    // REALIZAR LAS ACCIONES DE CONSULTA, INSERCIÓN, ACTUALIZACIÓN Y BORRADO

    // 5.- Cerrar la declaración y la conexión
    declaracion.close();
    conexion.close();
} catch (Exception e) {
    System.err.print(e.getMessage());
}
```

Imagen 5. Ejemplo de realizar una conexión a la base de datos

14.4.

Ejecución de consultas sobre la base de datos

Para cualquier tipo de consulta en una base de datos relacional se aplica el mismo patrón de desarrollo. Usando el objeto Statement se ejecutará la sentencia SQL con executeQuery(String) y los resultados serán tratados con el objeto ResultSet.



Imagen 6. Ejemplo de consulta sobre la tabla contactos

Con Statement se puede ejecutar cualquier tipo de sentencia de SQL que previamente se ha construido como una cadena de caracteres. Otra alternativa es usar PreparedStatement, permite construir una sentencia SQL dinámicamente, pasando parámetros a la consulta mediante métodos set.

```
try {
    Class.forName("org.mariadb.jdbc.Driver");
    String url = "jdbc:mariadb://localhost:3306/agenda";
    String usuario = "usuario";
    String contraseña = "1234";
    Connection conexion = DriverManager.getConnection(url, usuario, contraseña);

    String query = "SELECT * FROM contactos WHERE id=?";
    PreparedStatement declaracion = conexion.prepareStatement(query);
    declaracion.setInt(1, 615754236);
    ResultSet resultados = declaracion.executeQuery();

    System.out.println("id", " ", "nombre", " ", "apellido1", " ", " "
        + "apellido2", " ", "telefono", " ", "email");

    while (resultados.next()) {
        int id = resultados.getInt("id");
        String nombre = resultados.getString("nombre");
        String apellido1 = resultados.getString("apellido1");
        String apellido2 = resultados.getString("apellido2");
        int telefono = resultados.getInt("telefono");
        String email = resultados.getString("email");
        System.out.println(id + " " + nombre + " " + apellido1 + " "
            + apellido2 + " " + telefono + " " + email);
    }

    declaracion.close();
    conexion.close();
} catch (Exception e) {
    System.err.print(e.getMessage());
}
```

Imagen 7. Ejemplo de consulta sobre la tabla contactos con PreparedStatement



14.5.

Mecanismos de actualización de la base de datos

Las operación de inserción, actualización o borrado se realizan con transacciones empleando el método `executeUpdate()` del objeto `Statement`. Este método devuelve un valor de tipo numérico especificando el número de filas que se han visto afectados al ejecutarse.

Todas las transacciones que se ejecuten mediante `DriverManager` son autocommit, es decir, que se realizara commit automáticamente por cada transacción sin que se pueda cancelar. Si queremos cambiar este comportamiento se tendrá que indicar mediante el método `setAutoCommit(false)` y ya estaremos obligados a usar `commit()` y `rollback()` para confirmar o cancelar la transacciones.

INSERCIÓN

La inserción es la operación de añadir nuevas filas en las tablas de la base de datos, se usará la sentencia `INSERT INTO` de SQL.

```
Class.forName("org.mariadb.jdbc.Driver");
String url = "jdbc:mariadb://localhost:3306/agenda";
String usuario = "usuario";
String contraseña = "1234";
Connection conexion = DriverManager.getConnection (url, usuario, contraseña);

String query = "INSERT INTO CONTACTOS (nombre,apellido1,apellido2,telefono,email) VALUES (?,?, ?,?,?)";
PreparedStatement declaracion = conexion.prepareStatement(query);

declaracion.setString(1, "Pascual");
declaracion.setString(2, "Lozano");
declaracion.setString(3, "García");
declaracion.setInt(4, 615754236);
declaracion.setString(5, "pascual@email.com");
declaracion.executeUpdate();

declaracion.close();
conexion.close();
```

Es un valor ? por cada campo



Se trata de una query dinámica, podemos ver que se insertan/ modifican los valores a posteriori de haber determinado una query



Imagen 8. Ejemplo de inserción

ACTUALIZACIÓN

Se puede actualizar cualquier fila de la base de datos, usando la sentencia `UPDATE` de SQL

```
Class.forName("org.mariadb.jdbc.Driver");
String url = "jdbc:mariadb://localhost:3306/agenda";
String usuario = "usuario";
String contraseña = "1234";
Connection conexion = DriverManager.getConnection (url, usuario, contraseña);
Statement declaracion = conexion.createStatement();

String query = "UPDATE contactos SET email='pas.lo@email.com' WHERE id=5";

int registrosActualizados = declaracion.executeUpdate(query);
System.out.println(registrosActualizados);

declaracion.close();
conexion.close();
```

Vemos que es estática



Imagen 9. Ejemplo de actualización



BORRADO

Podemos borrar cualquier registro usando la sentencia DELETE de SQL.

```
Class.forName("org.mariadb.jdbc.Driver");
String url = "jdbc:mariadb://localhost:3306/agenda";
String usuario = "usuario";
String contraseña = "1234";
Connection conexion = DriverManager.getConnection(url, usuario, contraseña);
Statement declaracion = conexion.createStatement();

String query = "DELETE FROM contactos WHERE id=5";

int registrosActualizados = declaracion.executeUpdate(query);
System.out.println(registrosActualizados);

declaracion.close();
conexion.close();
```

Imagen 10. Ejemplo de borrado

El objeto Statement permite ejecutar lotes de sentencias para mejorar el rendimiento y que solo haya una sola llamada a la base de datos y no una por cada sentencia. Se usará addBatch(consulta) por cada sentencia y finalmente executeBatch() para lanzar todo el bloque. Para este tipo de funcionalidad hay que desactivar el autocommit para garantizar la seguridad en la base de datos.

```
try {
    Class.forName("org.mariadb.jdbc.Driver");
    String url = "jdbc:mariadb://localhost:3306/agenda";
    String usuario = "usuario";
    String contraseña = "1234";
    Connection conexion = DriverManager.getConnection(url, usuario, contraseña);
    conexion.setAutoCommit(false);

    Statement declaracion = conexion.createStatement();

    String query1 = "INSERT INTO CONTACTOS (nombre,apellidol,apellido2,telefono,email) "
        + "VALUES ('Amparo','Carrión','Gonzalez','656112233','email@email.com')";
    String query2 = "UPDATE contactos SET email='email@email.com' WHERE id=4";

    declaracion.addBatch(query1);
    declaracion.addBatch(query2);

    int[] registrosAfectados = declaracion.executeBatch();

    conexion.commit();

    declaracion.close();
    conexion.close();
} catch (Exception e) {
    System.err.print(e.getMessage());
}
```

Imagen 11. Ejemplo de ejecución por lotes



14.6.

Recuperación de la información

Cualquier base de datos tiene metainformación que es necesaria para su gestión, por ejemplo, los usuarios que tienen acceso a la base de datos o los esquemas y base de datos que contiene. Para conocer esta metainformación podemos hacer uso de sentencias SQL y ejecutarlas como hemos visto en el punto 14.4 o podemos hacer uso de DatabaseMetaData del objeto Connection.

```
try {
    Class.forName("org.mariadb.jdbc.Driver");
    String url = "jdbc:mariadb://localhost:3306/agenda";
    String usuario = "usuario";
    String contraseña = "1234";
    Connection conexion = DriverManager.getConnection (url, usuario, contraseña);

    // USO DE DATABASEMETADATA
    DatabaseMetaData metadatos = conexion.getMetaData();
    ResultSet resultados = metadatos.getCatalogs();

    System.out.println("- Obtener esquemas o base de datos:");
    while (resultados.next()) {
        String basededatos = resultados.getString("TABLE_CAT");
        System.out.println(basededatos);
    }

    // USO DIRECTO CON SQL
    Statement declaracion = conexion.createStatement();
    String query = "SELECT * FROM mysql.user";
    ResultSet res = declaracion.executeQuery(query);

    System.out.println("- Obtener usuarios de la base de datos:");
    while (res.next()) {
        String user = res.getString("USER");
        System.out.println(user);
    }

    declaracion.close();
    conexion.close();
} catch (Exception e) {
    System.err.print(e.getMessage());
}
```

- Obtener esquemas o base de datos:
agenda
bbddmysql
information_schema
mysql
performance_schema
sys
- Obtener usuarios de la base de datos:
mariadb.sys
root
root
root
root
usuario

Imagen 12. Consulta de metadatos

14.7.

Manipulación de la información

Igual que es posible realizar sentencias DML para manipular los datos de nuestras tablas, podemos realizar sentencia de DDL para definir los datos, crear base de datos, tablas, vistas, etc. el procedimiento a seguir es el mismo visto en el punto 14.5.

```
try {
    Class.forName("org.mariadb.jdbc.Driver");
    String url = "jdbc:mariadb://localhost:3306/agenda";
    String usuario = "usuario";
    String contraseña = "1234";
    Connection conexion = DriverManager.getConnection (url, usuario, contraseña);
    Statement declaracion = conexion.createStatement();

    String query = "CREATE TABLE Direcciones ("
        + "id INT AUTO_INCREMENT PRIMARY KEY,"
        + "via VARCHAR(50) NOT NULL,"
        + "nombrevia VARCHAR(200),"
        + "localidad VARCHAR(200),"
        + "codigopostal VARCHAR(10))";

    declaracion.executeUpdate(query);

    declaracion.close();
    conexion.close();
} catch (Exception e) {
    System.err.print(e.getMessage());
}
```

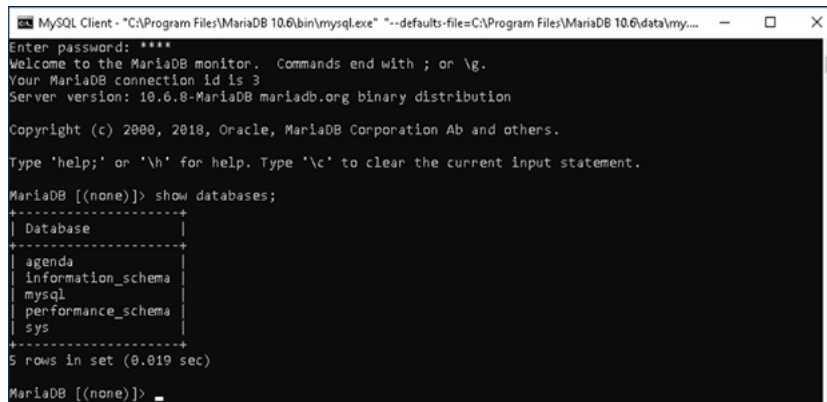
Imagen 13. Creación de una tabla

14.8.

Utilización de asistentes

En este tema hemos trabajado directamente desde código java para el acceso y manipulación de una base de datos, pero en ningún momento hemos comprobado los cambios realizados en la base de datos, simplemente teníamos la respuesta si se había ejecutado correctamente o no la sentencia.

Existen herramientas para operar directamente sobre una base de datos y que nos permitan consultar que se están haciendo correctamente los cambios desde código para verificar su funcionamiento. MariaDB contiene la herramienta **MySQL Client**, un cliente de tipo consola para introducir comandos SQL para operar con las bases de datos MySQL.



```

MySQL Client - "C:\Program Files\MariaDB 10.6\bin\mysql.exe" --defaults-file=C:\Program Files\MariaDB 10.6\data\my...
Enter password: ****
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.6.8-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| agenda   |
| information_schema |
| mysql    |
| performance_schema |
| sys      |
+-----+
5 rows in set (0.019 sec)

MariaDB [(none)]>
  
```

Imagen 14. Pantalla principal de MySQL Client

Los clientes a modo consola son herramientas sencillas, si queremos trabajar de forma gráfica tenemos otros clientes, como HeidiSQL exclusivo solo para MySQL o SQL Developer para diferentes tipos de bases de datos, por ejemplo, MySQL o Oracle.

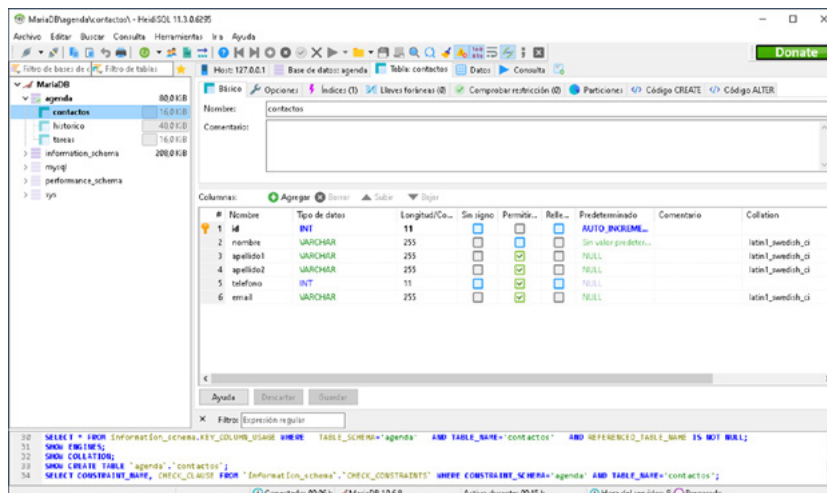


Imagen 15. Pantalla principal de HeidiSQL

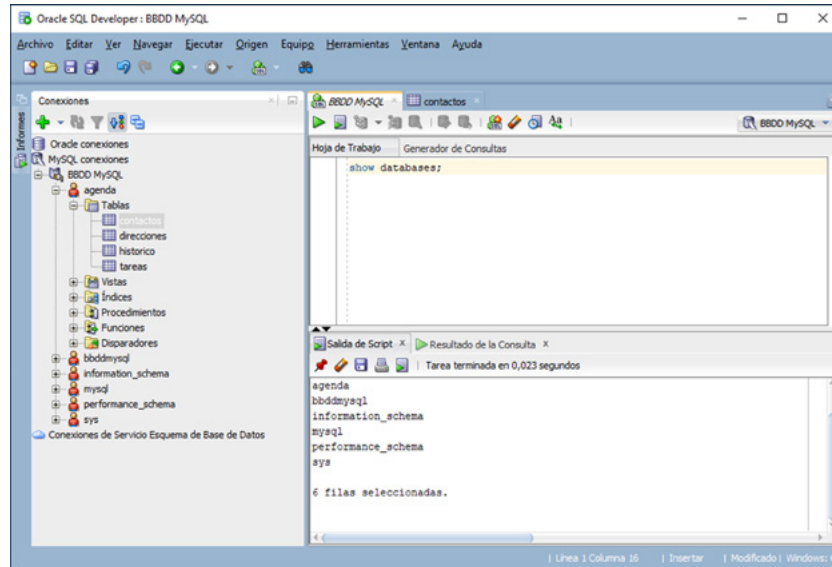


Imagen 16. Pantalla principal de SQL Developer

Los clientes gráficos tienen la característica principal de tener una interfaz gráfica o asistentes para realizar cualquier operación sobre la base de datos, ocultando todo el código y sentencias SQL para dar facilidad de uso y sin tener un conocimiento alto sobre SQL, aun así, deben de tener un apartado donde se pueda consultar las operaciones realizadas mediante código.



 www.universae.com

