

Manipular información de una base de datos

Manipular información de una base de datos	1
¿Qué aprenderás?	2
Introducción	2
Creación del modelo de datos	3
Importar Driver JDBC a Eclipse	4
Creación de clase de manipulación de datos	6
Sentencias DML	11
Ejecutar un insert	11
Ejecutar un delete	12
Ejecutar un delete masivo en cascada	13



¡Comencemos!

¿Qué aprenderás?

- Crear el modelo de datos.
- Vincular el driver ojdbc8.jar al proyecto.
- Programar la clase utilizando el api JDBC.
- Ejecutar un insert, delete, delete en cascada y update.

Introducción

Es tiempo de abrir el DBEAVER o SQL DEVELOPER para crear nuestras tablas para empezar a utilizar el driver JDBC y conectarnos a la base de datos.

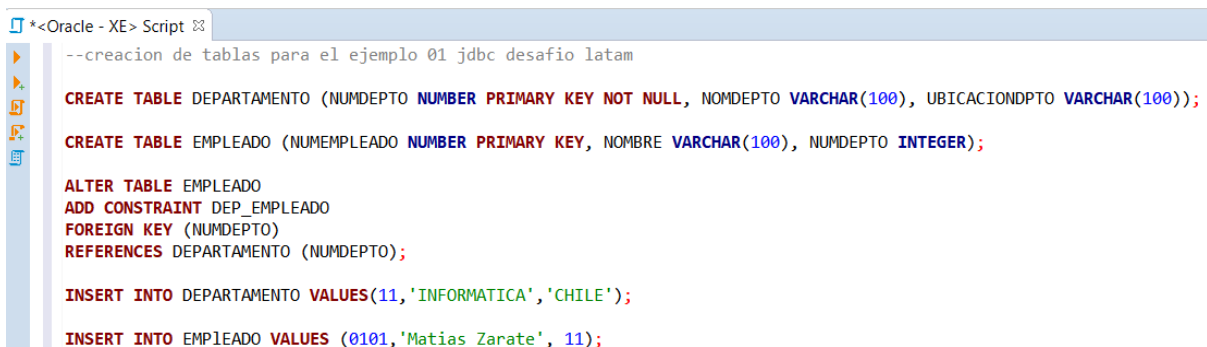
Este ejemplo consta de un programa de consola básico en java, que se conectará a una tabla de base de datos que crearemos para listar sus elementos en la aplicación.

Creación del modelo de datos

El siguiente script es utilizado para crear las dos tablas del ejemplo. El negocio indica que quiere registrar los departamentos de la empresa las cuales están repartidas en el mundo y a la vez mantener a los empleados y su departamento.

Para eso creamos dos tablas: DEPARTAMENTO y EMPLEADO.

```
CREATE TABLE DEPARTAMENTO (NUMDEPTO NUMBER PRIMARY KEY NOT NULL,  
NOMDEPTO VARCHAR(100), UBICACIONDPTO VARCHAR(100));  
  
CREATE TABLE EMPLEADO (NUMEMPLEADO NUMBER PRIMARY KEY, NOMBRE  
VARCHAR(100), NUMDEPTO INTEGER);  
  
ALTER TABLE EMPLEADO  
ADD CONSTRAINT DEP_EMPLEADO  
FOREIGN KEY (NUMDEPTO)  
REFERENCES DEPARTAMENTO (NUMDEPTO);  
  
INSERT INTO DEPARTAMENTO VALUES(11, 'INFORMATICA', 'CHILE');  
  
INSERT INTO EMPLEADO VALUES(0101, 'Matias Zarate', 11);
```



```
*<Oracle - XE> Script  
--creacion de tablas para el ejemplo 01 jdbc desafio latam  
  
CREATE TABLE DEPARTAMENTO (NUMDEPTO NUMBER PRIMARY KEY NOT NULL, NOMDEPTO VARCHAR(100), UBICACIONDPTO VARCHAR(100));  
  
CREATE TABLE EMPLEADO (NUMEMPLEADO NUMBER PRIMARY KEY, NOMBRE VARCHAR(100), NUMDEPTO INTEGER);  
  
ALTER TABLE EMPLEADO  
ADD CONSTRAINT DEP_EMPLEADO  
FOREIGN KEY (NUMDEPTO)  
REFERENCES DEPARTAMENTO (NUMDEPTO);  
  
INSERT INTO DEPARTAMENTO VALUES(11, 'INFORMATICA', 'CHILE');  
  
INSERT INTO EMPLEADO VALUES (0101, 'Matias Zarate', 11);
```

Imagen 1. Creando tablas departamento y empleado.

Fuente: Desafío Latam.

Importar Driver JDBC a Eclipse

Abrimos Eclipse y generamos un nuevo proyecto java normal. Para añadir el driver, simplemente seleccionamos con botón derecho sobre la opción *Referenced Libraries* y pinchamos la opción *Build Path, Configure Build Path*.

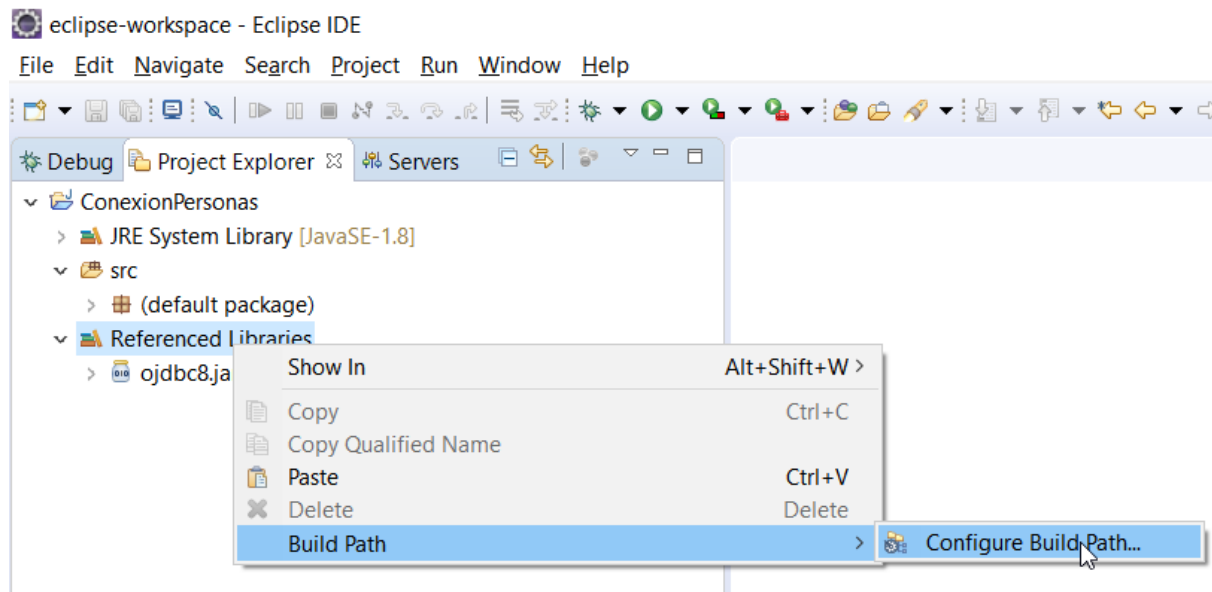


Imagen 2. Configure Build Path.
Fuente: Desafío Latam.

En la ventana Properties buscar la opción *Java Build Path*, y en la pestaña Libraries pinchar el botón *Add External Jar* para seleccionar el ojdbc8.jar.

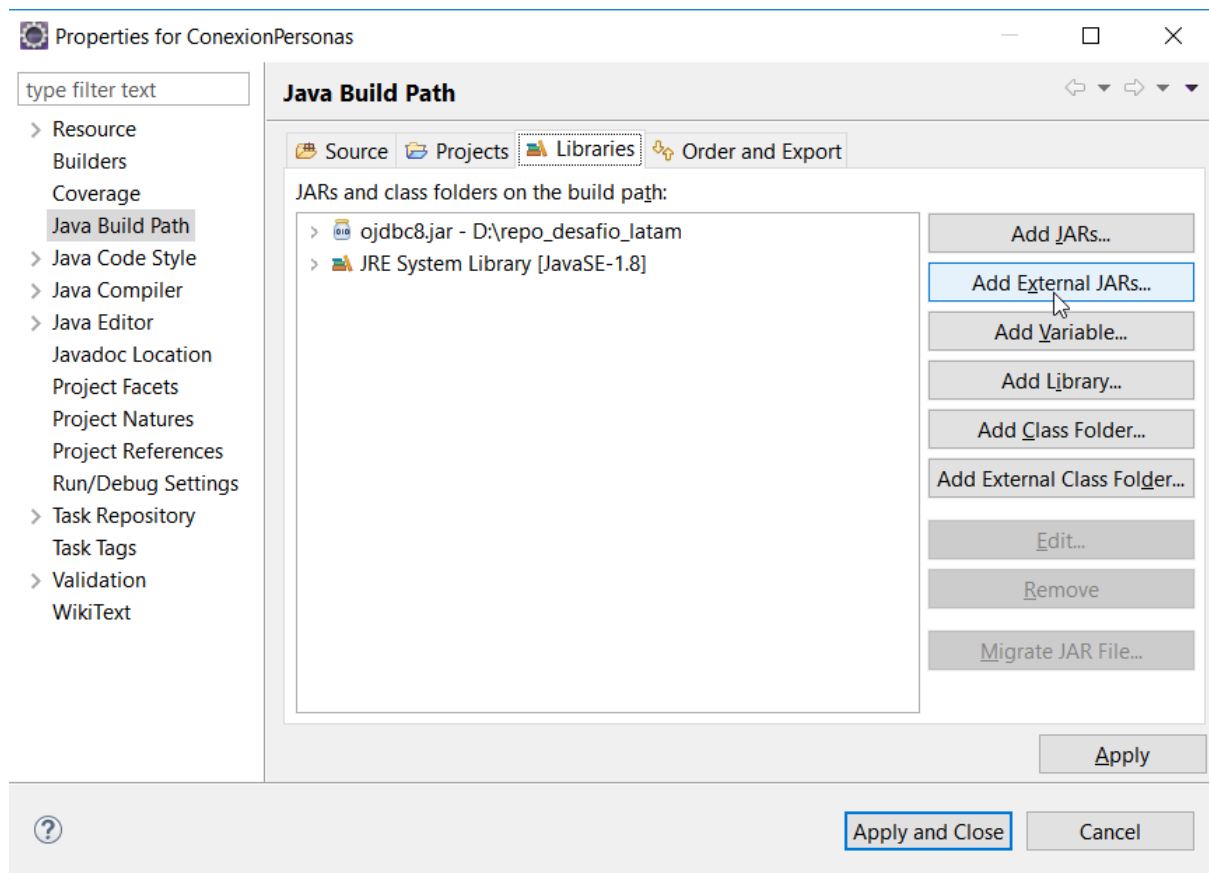


Imagen 3. Agregando .jar
Fuente: Desafío Latam.

Con estas acciones, el entorno está listo para poder generar la conexión entre la base de datos Oracle y el programa java. A continuación se expone la clase que utiliza el api JDBC.

Creación de clase de manipulación de datos

La creación de la clase se muestra en la imagen siguiente, en la cual se especifican los sectores de declaración de variables de uso *jdbc*, inicialización de conexión, driver y manipulación de datos.

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
public class DemoPersonas {
    public static void main(String...args) {
        String usr = "sys as sysdba";
        String pwd = "admin";
        String driver = "oracle.jdbc.driver.OracleDriver";
        String url =
"jdbc:oracle:thin:@//localhost:1521/desafio_ejemplo01";
        Connection conn = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try{
            Class.forName(driver);
            conn = DriverManager.getConnection(url, usr, pwd);
            String sql = "SELECT * FROM departamento, empleado";
            pstmt = conn.prepareStatement(sql);
            rs = pstmt.executeQuery();
            while(rs.next()) {
                System.out.println("DEPARTAMENTOS::");
                System.out.println(rs.getInt("NUMDEPTO"));
                System.out.println(rs.getString("NOMDEPTO"));
                System.out.println(rs.getString("UBICACIONDPTO"));
                System.out.println("EMPLEADOS::");
                System.out.println(rs.getInt("NUMEMPLEADO"));
                System.out.println(rs.getString("NOMBRE"));
                System.out.println(rs.getInt("NUMDEPTO"));
            }
        }catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

```
DemoPersonas.java
1 import java.sql.DriverManager;
5 public class DemoPersonas {
6     public static void main(String...args) {
7         String usr = "sys as sysdba";
8         String pwd = "admin";
9         String driver = "oracle.jdbc.driver.OracleDriver";
10        String url = "jdbc:oracle:thin:@//localhost:1521/desafio_ejemplo01";
11        Connection conn = null;
12        PreparedStatement pstmt = null;
13        ResultSet rs = null;
14        try {
15            //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
16            // 1 PARTE LEVANTAR EL DRIVER
17            //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
18            // levantamos el driver
19            Class.forName(driver);
20            //establecemos la conexion
21            conn = DriverManager.getConnection(url,usr,pwd);
22            //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
23            // 2 PARTE GENERACION DE QUERY
24            //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
25            //definimos la query
26            String sql = "SELECT * FROM departamento, empleado";
27            //preparamos sentencias que ejecutaremos
28            pstmt = conn.prepareStatement(sql);
29            //ejecutamos sentencia y obtenemos los resultados
30            rs = pstmt.executeQuery();
31            //recorremos los resultados
32            while(rs.next()) {
33                System.out.println("DEPARTAMENTOS::");
34                System.out.println(rs.getInt("NUMDEPTO"));
35                System.out.println(rs.getString("NOMDEPTO"));
36                System.out.println(rs.getString("UBICACIONDPTO"));
37                System.out.println("EMPLEADOS::");
38                System.out.println(rs.getInt("NUMEMPLEADO"));
39                System.out.println(rs.getString("NOMBRE"));
40                System.out.println(rs.getInt("NUMDEPTO"));
41            }
42        } catch (Exception ex) {
43            ex.printStackTrace();
44        }
45    }
46 }
```

Imagen 4. Clase de manipulación de datos.

Fuente: Desafío Latam.

Los import cuentan con:

```
1 import java.sql.DriverManager;
2 import java.sql.Connection;
3 import java.sql.PreparedStatement;
4 import java.sql.ResultSet;
```

Imagen 5. Imports.

Fuente: Desafío Latam.

Son las clases propias de JDBC que nos permite conectarnos a las bases de datos. Los nombres son autoexplicativos.

- **DriverManager**: Se encarga de administrar la carga del driver de conexión a base de datos.
- **Connection**: Disponibiliza las clases utilizadas para generar y administrar la conexión a base de datos.
- **PreparedStatement**: Ofrece clases para generar las sentencias con sus valores a utilizar.
- **ResultSet**: Elemento que nos otorga los datos rescatados desde Oracle. Mediante resultset podemos acceder a ellos y manipularlos.

Analizaremos el código anteriormente expuesto.

Líneas 7 al 10: Se declaran las variables con datos de conexión para que la aplicación se pueda conectar a las bases de datos.

```
7 | String usr = "sys as sysdba";
8   String pwd = "admin";
9   String driver = "oracle.jdbc.driver.OracleDriver";
10  String url = "jdbc:oracle:thin:@//localhost:1521/desafio_ejemplo01";
11  Connection conn = null;
12  PreparedStatement pstmt = null;
13  ResultSet rs = null;
```

Imagen 6. Analizando el código entre las líneas 7 a la 13.

Fuente: Desafío Latam.

- **usr**: Corresponde al nombre de usuario del esquema de base de datos. En este caso utilizamos el usuario sys.
- **pwd**: Configuración de la password de oracle.
- **driver**: La ruta del driver que se está utilizando. Pueden buscar en internet si se necesita otro motor de base de datos.
- **url**: String de conexión necesario para poder acceder a los recursos. Utiliza la tecnología thin.

Líneas 11 a la 13: Declaración de instancias de las clases *Connection*, *Prepared Statement* y *ResultSet*. Todas son declaradas pero no inicializadas.


```
11      Connection conn = null;  
12      PreparedStatement pstmt = null;  
13      ResultSet rs = null;
```

Imagen 7. Declaración de las instancias Connection, Prepared Statement y Resultset.
Fuente: Desafío Latam.

Líneas 14 a 22: Iniciamos el driver de conexión mediante *Class.forName(driver)*. Driver fue declarado más arriba. Luego, utilizando la instancia de *Connection* usamos la clase *DriverManager.getConnection* que recibe como parámetros la *url*, el usuario y el password de la base de datos.

```
14      try {  
15          //:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::  
16          // 1 PARTE LEVANTAR EL DRIVER  
17          //:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::  
18          // levantamos el driver  
19          Class.forName(driver);  
20          //establecemos la conexion  
21          conn = DriverManager.getConnection(url,usr,pwd);
```

Imagen 8. Analizando código entre las líneas 14 a la 22.
Fuente: Desafío Latam.

Líneas 23 en adelante: Se puede ver que ya generando y estableciendo la conexión es posible generar la sentencia sql que queramos, la cual es pasada como parámetro en la línea 28 a la instancia de conexión y su método *prepareStatement*.

La instancia de *resultset* en la línea 30 se encarga de recibir los datos que retorna la consulta y si es efectiva la ejecución, se recorre tal *resultset* en el ciclo *while*. Con estos pasos es posible acceder a los datos de la sentencia mediante *rs.getInt* y el nombre del campo de base de datos.

Notar que si se quiere rescatar un valor numérico se utiliza el método *getInt*, si el dato es un string se utiliza *getString*.

```
22 //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
23 // 2 PARTE GENERACION DE QUERY
24 //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
25 //definimos la query
26 String sql = "";
27 sql+="INSERT INTO DEPARTAMENTO (NUMDEPTO,NOMDEPTO,UBICACIONDPTO)";
28 sql+="VALUES(?,?,?)";
29 //preparamos sentencias que ejecutaremos
30 pstmt = conn.prepareStatement(sql);
31 //adjuntamos los valores a los parametros
32 pstmt.setInt(1,12);
33 pstmt.setString(2, "CONTABILIDAD");
34 pstmt.setString(3, "MEXICO");
35 int resultado = pstmt.executeUpdate();
36 //si la variable resultado es mayor a 0 el insert fue correcto
37 if(resultado > 0) {
38     System.out.println("Fila correctamente insertada !");
39 }else {
40     System.out.println("Ocurrio un error insertando el departamento");
41 }
```

Imagen 9. Generación de Query.

Fuente: Desafío Latam.

La salida de la sentencia es la siguiente:

```
Console Problems Debug Shell
<terminated> DemoPersonas [Java Application] C:\Program Files\Java\jre1.8.0_211\bin\javaw.exe (Jul 17, 2019, 6:42:03 PM)
DEPARTAMENTOS:::
11
INFORMATICA
CHILE
EMPLEADOS:::
101
Matias Zarate
11
```

Imagen 10. Resultado en la consola.

Fuente: Desafío Latam.

Es importante cerrar las instancias de clases luego de utilizarlas. Esto dentro de un bloque *finally*:

```
44 }finally {
45     //cerramos todos los recursos en orden inverso al que fueron declarados
46     try {
47         if(rs != null) rs.close();
48         if(pstmt!=null) pstmt.close();
49         if(conn!=null) conn.close();
50     } catch (Exception e) {
51         e.printStackTrace();
52     }
53 }
```

Imagen 11. Bloque finally.

Fuente: Desafío Latam.

Sentencias DML

Ejecutar un insert

Son conocidas como sentencias de modificación de datos. Entre ellas están los INSERT, DELETE, UPDATES.

A continuación veremos el procedimiento para generar un insert a la tabla de departamentos. Utilizaremos la misma clase generada anteriormente, así que para trabajar más rápido puedes copiar dicha clase y eliminar las sentencias de *select*.

```
22 //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
23 // 2 PARTE GENERACION DE QUERY
24 //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
25 //definimos la query
26 String sql = "";
27 sql+="INSERT INTO DEPARTAMENTO (NUMDEPTO,NOMDEPTO,UBICACIONDPTO)";
28 sql+="VALUES(?,?,?)";
29 //preparamos sentencias que ejecutaremos
30 pstmt = conn.prepareStatement(sql);
31 //adjuntamos los valores a los parametros
32 pstmt.setInt(1,12);
33 pstmt.setString(2, "CONTABILIDAD");
34 pstmt.setString(3, "MEXICO");
35 int resultado = pstmt.executeUpdate();
36 //si la variable resultado es mayor a 0 el insert fue correcto
37 if(resultado > 0) {
38     System.out.println("Fila correctamente insertada !");
39 }else {
40     System.out.println("Ocurrio un error insertando el departamento");
41 }
```

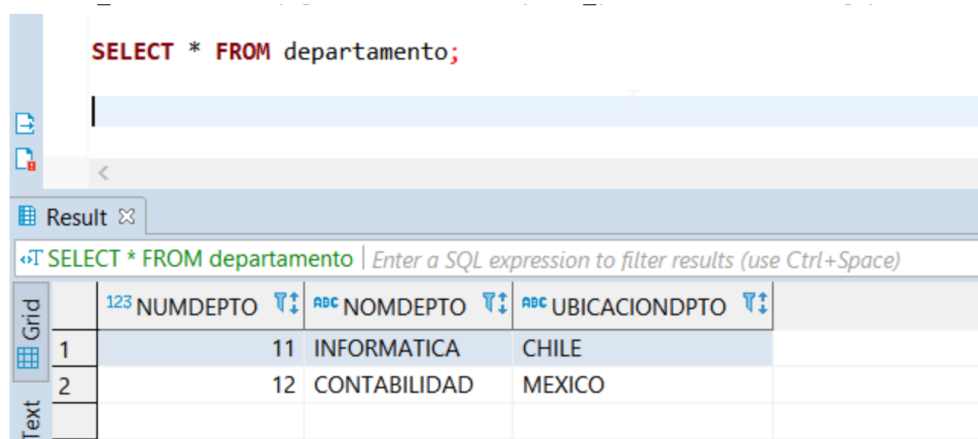
Imagen 12 Generando consultas.

Fuente: Desafío Latam.

La línea 28 cuenta con la sentencia VALUES y una serie de símbolos de interrogación. Indica que cada uno de los símbolos es un valor a insertar en la tabla, y gracias a la variable pstmt seteamos los valores de acuerdo al orden de los mismos parámetros de la tabla.

Fijarse que el primer valor de la tabla DEPARTAMENTOS es el id, el segundo valor es el nombre del departamento y el tercer valor es la ubicación. El método de pstmt recibe la ubicación y el nombre del valor.

Para ejecutar una sentencia DML se utiliza el método `executeUpdate`, el cual genera la transacción y modifica la tabla. Este método retorna un entero mayor a 0 si la ejecución fue correcta. Luego de ejecutar el código, la tabla ya tiene los nuevos registros:



The screenshot shows a database application window. At the top, a text area contains the SQL query: `SELECT * FROM departamento;`. Below the text area is a tab labeled 'Result'. Under the 'Result' tab, there is a search bar with the text 'SELECT * FROM departamento' and a hint 'Enter a SQL expression to filter results (use Ctrl+Space)'. Below the search bar is a table with three columns: 'NUMDEPTO', 'NOMDEPTO', and 'UBICACIONDPTO'. The table has two rows of data.

	NUMDEPTO	NOMDEPTO	UBICACIONDPTO
1	11	INFORMATICA	CHILE
2	12	CONTABILIDAD	MEXICO

Imagen 13. Nuevos registros.
Fuente: Desafío Latam.

Ejecutar un delete

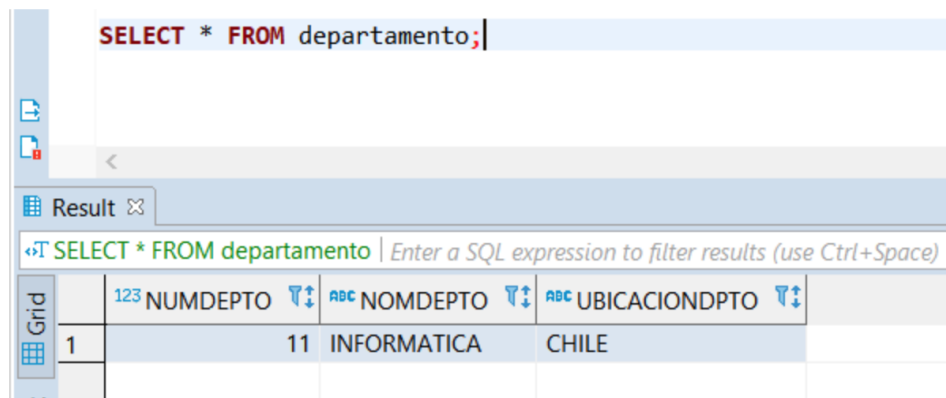
Eliminaremos el mismo registro que se insertó en la clase anterior. Para ello, se presenta el siguiente código:

```
25         //definimos la query
26         String sql = "";
27         sql+="DELETE FROM DEPARTAMENTO WHERE NUMDEPTO = ?";
28         //preparamos sentencias que ejecutaremos
29         pstmt = conn.prepareStatement(sql);
30         //adjuntamos los valores a los parametros
31         pstmt.setInt(1,12);
32         int resultado = pstmt.executeUpdate();
33         //si la variable resultado es mayor a 0 el insert fue correcto
34         if(resultado == 1) {
35             System.out.println("Fila correctamente eliminada !");
36         }else {
37             System.out.println("Ocurrio un error eliminando el departamento");
38         }
```

Imagen 14. Eliminando registro.
Fuente: Desafío Latam.

La sentencia `sql` cambia a un delete con su correspondiente *where* en donde indicamos que elemento se quiere eliminar. Como se puede ver la mecánica es la misma que en el *insert*.

Luego de la ejecución del programa, el registro es eliminado.



The screenshot shows a SQL IDE interface. At the top, a query editor contains the text `SELECT * FROM departamento;`. Below the editor, a tab labeled 'Result' is active. The results pane shows the execution of the query, displaying a table with the following data:

	123 NUMDEPTO	ABC NOMDEPTO	ABC UBICACIONDPTO
1	11	INFORMATICA	CHILE

Imagen 15. Elemento eliminado.
Fuente: Desafío Latam.

Ejecutar un delete masivo en cascada

Hay ocasiones en que se necesitan eliminar datos de tablas que tienen relación entre sí, por ejemplo si el gerente general de la empresa decide que el departamento de recursos humanos va a desaparecer y junto a él todos sus empleados (los empleados serán reubicados obviamente) podemos generar un delete en cascada que eliminará todos los departamentos de recursos humanos y luego a todos los empleados que pertenezcan a dicha área.

Recordemos que la tabla DEPARTAMENTOS y la tabla EMPLEADOS tienen una relación directa entre el id de departamento de cada una lo que significa que un empleado está asociado a un departamento en especial.

Para poder eliminar en cascada es necesario añadir la sentencia `ON DELETE CASCADE` la declaración del *foreign key*:

```
ALTER TABLE EMPLEADO
ADD CONSTRAINT DEP_EMPLEADO
FOREIGN KEY (NUMDEPTO)
REFERENCES DEPARTAMENTO (NUMDEPTO) ON DELETE CASCADE;
```

Para eliminar al departamento de RRHH junto a todos sus empleados, simplemente debemos generar la sentencia de delete en contra de la tabla principal EMPLEADO en donde el id de departamento sea el identificador del departamento de recursos humanos.

```
DELETE FROM EMPLEADO  
AWHERE A.NUMDEPTO = 11;
```

La sentencia ON DELETE CASCADE es quien se encarga de ejecutar el borrado masivo dependiendo de la relación a la clave foránea especificada.

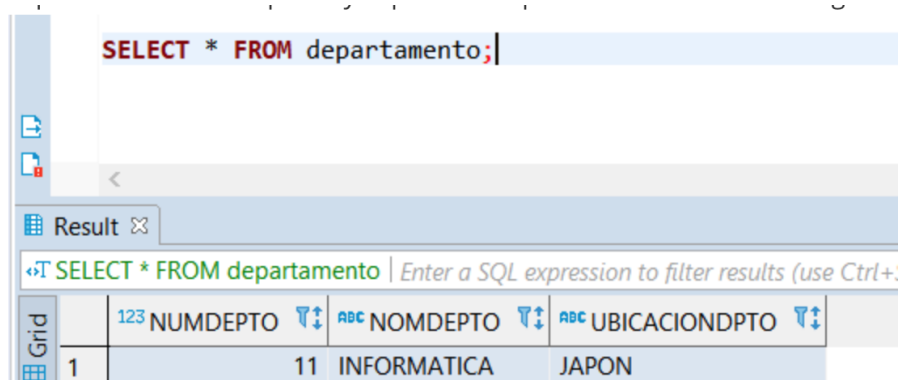
Al ejecutar un update para finalizar, se quiere cambiar el país del departamento 11 a Japón. Para ello, implementamos la última sentencia.

```
27 //definimos la query  
28 String sql = "";  
29 sql+="UPDATE DEPARTAMENTO SET UBICACIONDPTO = ? ";  
30 //preparamos sentencias que ejecutaremos  
31 pstmt = conn.prepareStatement(sql);  
32 //adjuntamos los valores a los parametros  
33 pstmt.setString(1,"JAPON");  
34 int resultado = pstmt.executeUpdate();  
35 System.out.println(resultado + "filas actualizadas");
```

Imagen 16. Realizando un update.

Fuente: Desafío Latam.

Con la ejecución de la clase, se actualiza el valor del país de ubicación a japon. En el código se ve la implementación del update y la pasada de parámetros se mantiene igual.



The screenshot shows a database application interface. At the top, a SQL query is entered in a text field: `SELECT * FROM departamento;`. Below the query field, there is a 'Result' tab. Under this tab, the same query is displayed, followed by a table of results. The table has three columns: 'NUMDEPTO', 'NOMDEPTO', and 'UBICACIONDPTO'. The first row of data shows the department with ID 11, named 'INFORMATICA', located in 'JAPON'.

	NUMDEPTO	NOMDEPTO	UBICACIONDPTO
1	11	INFORMATICA	JAPON

Imagen 17. Resultado de la ejecución del update.

Fuente: Desafío Latam.