Acceso a Datos

UT3. MANEJO DE CONECTORES
BASES DE DATOS RELACIONALES
PROCEDIMIENTOS

1. Ejecución de procedimientos

- Los procedimientos almacenados en la base de datos consisten en un conjunto de sentencias SQL que se pueden llamar por su nombre para llevar a cabo alguna tarea en la base de datos.
- Pueden definirse con parámetros de entrada (IN), de salida (OUT), de entrada/salida (INOUT) o sin ningún parámetro.
- También pueden devolver un valor, en este caso se trataría de una función.
- Las técnicas para desarrollar procedimientos y funciones almacenadas dependen del sistema gestor de base de datos.

1. Ejecución de procedimientos

- El siguiente ejemplo muestra un procedimiento de nombre **subida_sal** en MySQL que sube el sueldo a los empleados de un departamento, el procedimiento recibe dos parámetros de entrada que son el número de departamento (**d**) y la subida (**subida**):

```
DELIMITER //
CREATE PROCEDURE subida_sal (d INT, subida INT)
BEGIN
UPDATE empleados SET salario = salario + subida WHERE dept_no=d;
COMMIT;
END//
```

1. Ejecución de procedimientos

- La sentencia **DELIMITER** cambia el carácter de terminación ';' por cualquier otro carácter, en este caso elegimos '//'. Se hace con el fin de que MySQL no termine el procedimiento al encontrar el primer punto y coma.

- Para llamar al procedimiento subida_sal sería: `CALL subida_sal(10,20)`

1. Ejecución de procedimientos

- La interfaz **CallableStatement** permite que se pueda llamar desde Java a los procedimientos almacenados, para crear un objeto se llama al método **prepareCall(String)** del objeto **Connection**, el siguiente ejemplo declara la llamada al procedimiento **subida_sal** que tiene dos parámetros y para darles valor se utilizan los marcadores de posición (?):

```
String sql= "{ call subida_sal (?, ?) } ";
CallableStatement llamada = conexion.prepareCall(sql);
```

1. Ejecución de procedimientos

- Hay cuatro formas de declarar las llamadas a los procedimientos y funciones que dependen del uso u omisión de parámetros, y de la devolución de valores. Son las siguientes:

- {call procedimiento}: para un procedimiento almacenado sin parámetros.
- {? = call función }: para una función almacenada que devuelve un valor y no recibe parámetros, el valor se recibe a la izquierda del igual y es el primer parámetro.
- {call procedimiento(?, ?,...) }: para un procedimiento almacenado que recibe parámetros.
- { ? = call función(?, ?,...) }: para una función almacenada que devuelve un valor (primer parámetro) y recibe varios parámetros.

```java
public static void crearProcedimientoSalario() {
    try {
        // Cargar el driver
        Class.forName("com.mysql.cj.jdbc.Driver");

        // Establecemos la conexión con la BD
        Connection conexion =
                DriverManager.getConnection("jdbc:mysql://localhost/empresa", "root", "");

        Statement statement = conexion.createStatement();

        String queryDrop = "DROP PROCEDURE IF EXISTS subida_sal;";

        //String queryCreate = "DELIMITER // ";
        String queryCreate = "CREATE PROCEDURE subida_sal (IN d INT, IN subida INT) ";
        queryCreate += "BEGIN ";
        queryCreate += "UPDATE empleados SET salario = salario + subida WHERE dept_no=d; ";
        queryCreate += "END";
        // queryCreate += "DELIMITER ;";

        System.out.println(queryCreate);

        // drops the existing procedure if exists
        statement.execute(queryDrop);

        // then creates a new stored procedure
        statement.execute(queryCreate);

        System.out.println("Procedimiento creado");

        statement.close(); // Cerrar CallableStatement
        conexion.close(); // Cerrar conexión
    } catch (ClassNotFoundException en) {
        en.printStackTrace();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

1. Ejecución de procedimientos

```
public static void SubidaSalarioEmpleados(int dep, double subida) {
    try {
        // Cargar el driver
        Class.forName("com.mysql.cj.jdbc.Driver");
        // Establecemos la conexión con la BD
        Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/empresa", "root", "");

        String sql = "{call subida_sal(?, ?)}";

        // Preparamos la llamada
        CallableStatement cst = conexion.prepareCall(sql);

        // Damos valor a los argumentos
        cst.setInt(1, dep); // primer argumento-dep
        cst.setDouble(2, subida); // segundo argumento-subida
        cst.executeUpdate(); // ejecutar el procedimiento

        System.out.println("Subida realizada....");
        cst.close(); // Cerrar CallableStatement
        conexion.close(); // Cerrar conexión
    } catch (ClassNotFoundException en) {
        en.printStackTrace();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

1. Ejecución de procedimientos

- Si tenemos un procedimiento para mostrar datos, el código cambia ligeramente para recorrer el resultado.

- Por ejemplo, con el siguiente procedimiento en MySQL:

```
CREATE OR REPLACE procedure
getDepartamentos()
BEGIN
SELECT * FROM DEPARTAMENTOS;
END
```

```
public static void crearProcedimientoDepartamentos() {
    try {
        // Cargar el driver
        Class.forName("com.mysql.cj.jdbc.Driver");

        // Establecemos la conexión con la BD
        Connection conexion =
                DriverManager.getConnection("jdbc:mysql://localhost/empresa", "root", "");

        Statement statement = conexion.createStatement();

        String queryDrop = "DROP PROCEDURE IF EXISTS get_departamentos;";

        String queryCreate = "CREATE OR REPLACE PROCEDURE get_departamentos () ";
        queryCreate += "BEGIN ";
        queryCreate += "SELECT * FROM departamentos; ";
        queryCreate += "END";

        System.out.println(queryCreate);

        // drops the existing procedure if exists
        statement.execute(queryDrop);

        // then creates a new stored procedure
        statement.execute(queryCreate);

        System.out.println("Procedimiento creado");

        statement.close(); // Cerrar CallableStatement
        conexion.close(); // Cerrar conexión
    } catch (ClassNotFoundException en) {
        en.printStackTrace();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

1. Ejecución de procedimientos

```
public static void getDepartamentosMySQL() {
    try {
        // Cargar el driver
        Class.forName("com.mysql.cj.jdbc.Driver");

        // Establecemos la conexión con la BD
        Connection conexion =
                DriverManager.getConnection("jdbc:mysql://localhost/empresa", "root", "");
        String sql = "{call get_departamentos()}";

        // Preparamos la llamada
        CallableStatement cst = conexion.prepareCall(sql);
        ResultSet rs = cst.executeQuery(); // ejecutar el procedimiento

        while (rs.next()) {
            System.out.println("Codigo:" + rs.getString(1));
            System.out.println("Departamento: " + rs.getString(2));
        }
        cst.close();
        rs.close();
        conexion.close();
    } catch (ClassNotFoundException en) {
        en.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

2. Parámetros de salida

- Cuando un procedimiento o función tiene parámetros de salida (OUT) deben ser registrados antes de que la llamada tenga lugar, si no se registra se producirá un error.
- El método que se utilizará es: registerOutParameter(índice, tipoJDBC), el primer parámetro es la posición y el siguiente es una constante definida en la clase java.sql.Types.
- La clase Types define una constante para cada tipo genérico SQL, algunas son: TINYINT, SMALLINT, INTEGER, FLOAT, REAL, DOUBLE, NUMERIC,…..
- Por ejemplo, si el segundo parámetro de un procedimiento es OUT y de tipo VARCHAR habría que añadir:

```
cst.registerOutParameter(2, java.sql.Types.VARCHAR);
```

2. Parámetros de salida

- Una vez ejecutada la llamada al procedimiento, los valores de los parámetros OUT e INOUT se obtienen con los métodos getXXX(índice) similares a los utilizados para obtener los valores de las columnas en un Resultset.

```java
public static void crearProcedimientoLocalidad() {
    try {
        // Cargar el driver
        Class.forName("com.mysql.cj.jdbc.Driver");

        // Establecemos la conexión con la BD
        Connection conexion =
                DriverManager.getConnection("jdbc:mysql://localhost/empresa", "root", "");

        Statement statement = conexion.createStatement();

        String queryDrop = "DROP PROCEDURE IF EXISTS localidad_depart;";

        String queryCreate = "CREATE PROCEDURE localidad_depart (IN d INT, OUT localidad VARCHAR(15)) ";
        queryCreate += "BEGIN ";
        queryCreate += "SELECT loc INTO localidad FROM departamentos WHERE dept_no=d; ";
        queryCreate += "END";

        System.out.println(queryCreate);

        // drops the existing procedure if exists
        statement.execute(queryDrop);

        // then creates a new stored procedure
        statement.execute(queryCreate);

        System.out.println("Procedimiento creado");

        statement.close(); // Cerrar CallableStatement
        conexion.close(); // Cerrar conexión
    } catch (ClassNotFoundException en) {
        en.printStackTrace();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```java
public static void obtenerLocalidad(int dep) {
    try {
        // Cargar el driver
        Class.forName("com.mysql.cj.jdbc.Driver");
        // Establecemos la conexión con la BD
        Connection conexion =
                DriverManager.getConnection("jdbc:mysql://localhost/empresa", "root", "");

        String sql = "{call localidad_depart(?, ?)}";

        // Preparamos la llamada
        CallableStatement cst = conexion.prepareCall(sql);

        // Damos valor a los argumentos
        cst.setInt(1, dep); // primer argumento-dep

        cst.registerOutParameter(2, Types.VARCHAR);// devuelve la localidad

        cst.executeUpdate(); // ejecutar el procedimiento

        System.out.println("Localidad: " + cst.getString(2));

        cst.close(); // Cerrar CallableStatement
        conexion.close(); // Cerrar conexión
    } catch (ClassNotFoundException en) {
        en.printStackTrace();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

3. Ejemplos

- Más ejemplos sobre llamadas a procedimientos con JDBC:

https://www.codejava.net/java-se/jdbc/jdbc-examples-for-calling-stored-procedures-mysql

Dudas y preguntas