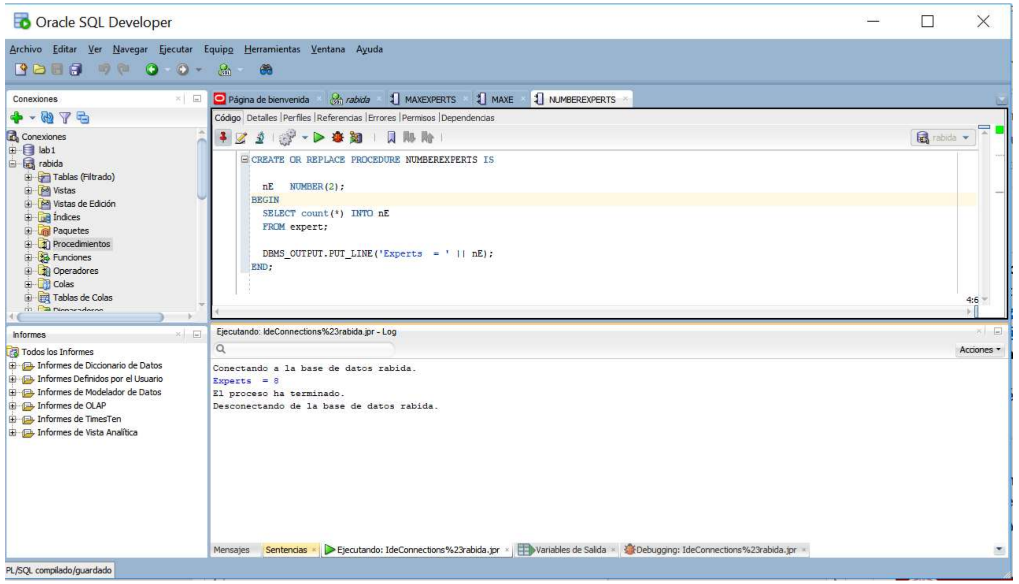
Information Systems Design and Development

Using Stored Procedures

Using Stored Procedures (and functions)

- A stored procedure is a group of SQL statements that form a logical unit and perform a particular task, and they are used to encapsulate a set of operations or queries to execute on a database server.
- Stored procedures can be compiled and executed with different parameters and results, and they can have any combination of input, output, and input/output parameters.
- Stored procedures are supported by most DBMSs, but there is a fair amount of variation in their syntax and capabilities

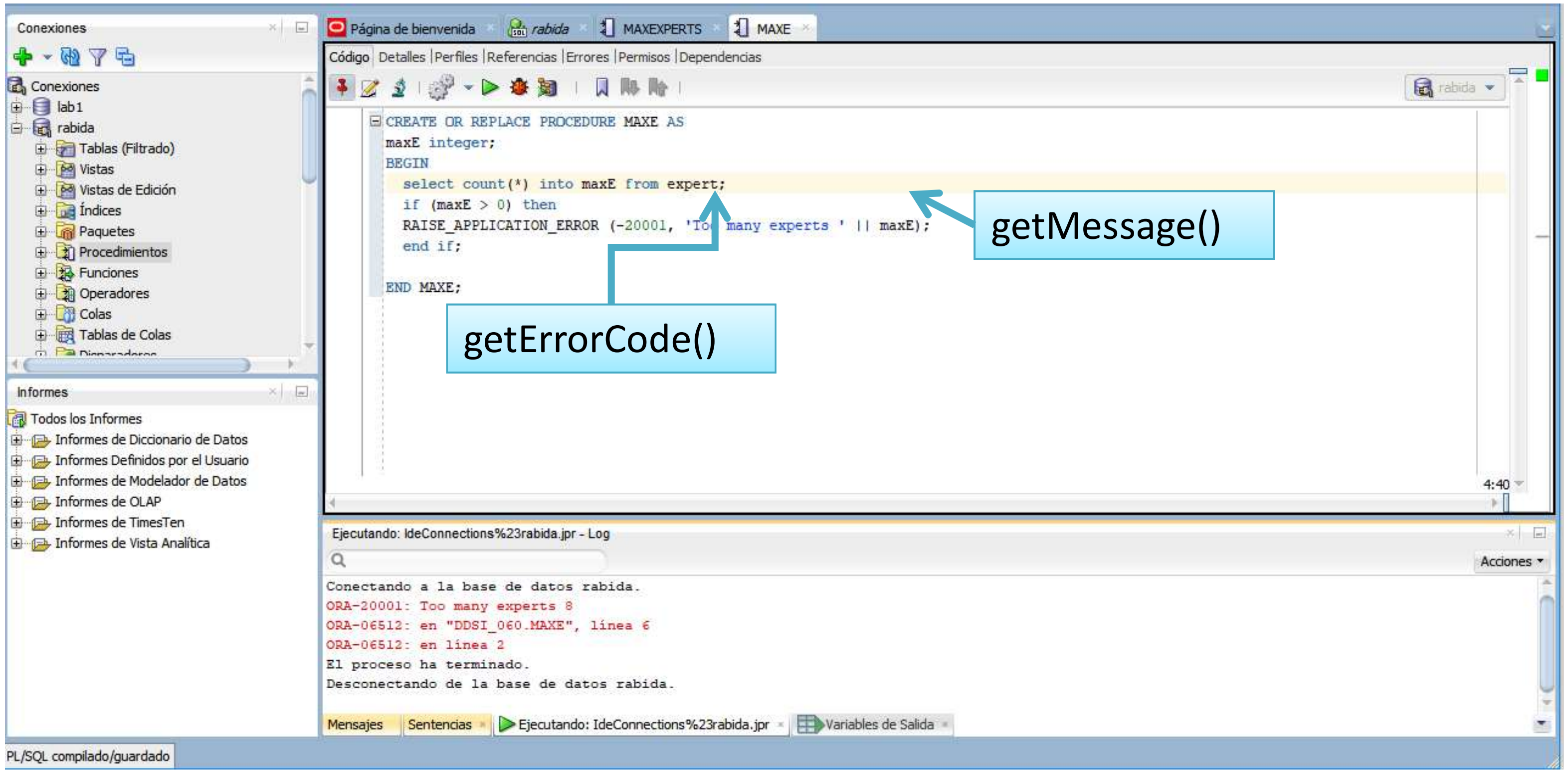

Oracle SQL Developer
Archivo Editar Ver Navegar Ejecutar Equipo Herramientas Ventana Ayuda
Conexiones
Conexiones
lab1
rabida
Tablas (Filtrado)
Vistas
Vistas de Edición
Índices
Paquetes
Procedimientos
Funciones
Operadores
Colas
Tablas de Colas
Informes
Todos los Informes
Informes de Diccionario de Datos
Informes Definidos por el Usuario
Informes de Modelador de Datos
Informes de OLAP
Informes de TimesTen
Informes de Vista Analítica
Página de bienvenida
rabida
MAXEXPERTS
MAXE
NUMBEREXPERTS
Código Detalles Perfiles Referencias Errores Permisos Dependencias
4:6
Ejecutando: IdeConnections%23rabida.jpr - Log
Acciones
Mensajes
Sentencias
Ejecutando: IdeConnections%23rabida.jpr
Variables de Salida
Debugging: IdeConnections%23rabida.jpr
PL/SQL compilado/guardado

```
CREATE OR REPLACE PROCEDURE NUMBEREXPERTS IS

  nE   NUMBER(2);
BEGIN
  SELECT count(*) INTO nE
  FROM expert;

  DBMS_OUTPUT.PUT_LINE('Experts  = ' || nE);
END;
```

```
Conectando a la base de datos rabida.
Experts  = 8
El proceso ha terminado.
Desconectando de la base de datos rabida.
```

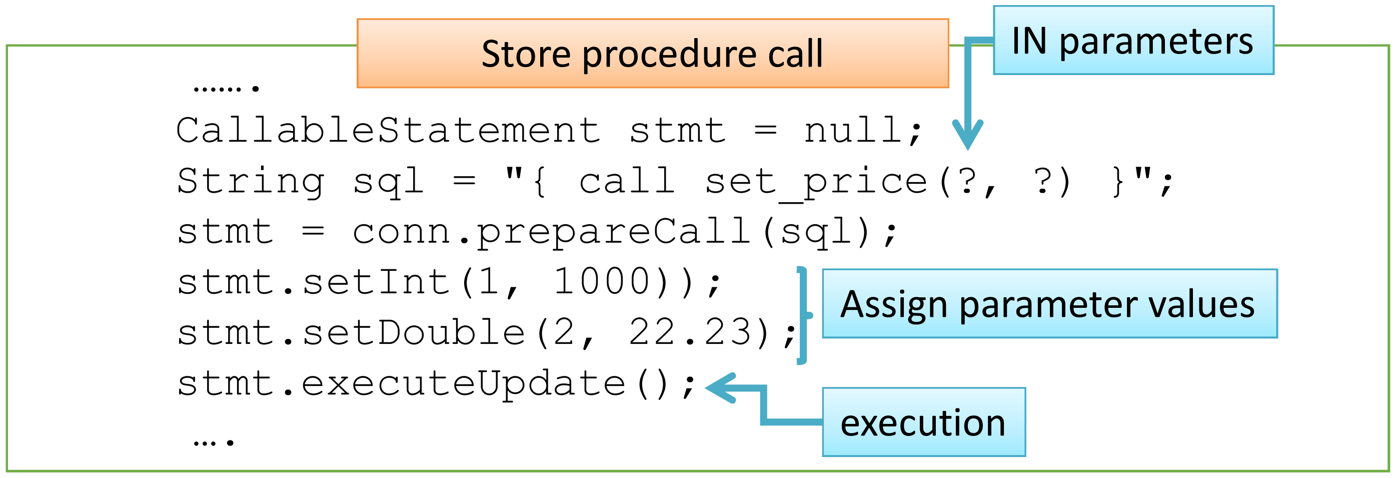
https://docs.oracle.com/cd/B28359_01/appdev.111/b28843/tdddg_procedures.htm

- These procedures can be used to perform calculations and operations.
- It is a good strategy to throw exceptions from stored procedures when a business logic error is encountered.
- This exception must send the error message so that it is captured in the Java program and treated in the class that calls the stored procedure



```
CREATE OR REPLACE PROCEDURE MAXE AS
maxE integer;
BEGIN
  select count(*) into maxE from expert;
  if (maxE > 0) then
  RAISE_APPLICATION_ERROR (-20001, 'Too many experts ' || maxE);
  end if;

END MAXE;
```

```
Conectando a la base de datos rabida.
ORA-20001: Too many experts 8
ORA-06512: en "DDSI_060.MAXE", línea 6
ORA-06512: en línea 2
El proceso ha terminado.
Desconectando de la base de datos rabida.
```



```
 …….
CallableStatement stmt = null;
String sql = "{ call set_price(?, ?) }";
stmt = conn.prepareCall(sql);
stmt.setInt(1, 1000));
stmt.setDouble(2, 22.23);
stmt.executeUpdate();
 ….
```

- To call stored procedures or stored functions in MySQL from JDBC, you use CallableStatement object.
- First, open a connection to MySQL server by creating a new Connection object.
- Then, prepare a stored procedure call and create a CallableStatement object by calling prepareCall() method of the Connection object.
- Next, pass all the parameters to the stored procedure. In this case, the set_price stored procedure accepts only one IN parameter.
- After that, execute the stored procedure by calling the executeUpdate() method of the CallableStatement object. It it returns a resultset (not this case), traverse the ResultSet to display the results.

- A stored procedure can have input parameters and output parameters, which are used to return information from the program. Especially, the functions always have an output parameter that is the value that the function returns.
- For a stored procedure or a function to return information using the output parameters, it is necessary to indicate, previously, what type each parameter is by means of the registerOutParameter () method.
- To retrieve the values of the output parameters we use the getXXX () method, where 'XXX' indicates the data type of the output parameter

Function call

```
….
CallableStatement stmt = null;
String sql = "{ ? = call get_price(?) }";
stmt = conn.prepareCall(sql);
stmt.setInt(2, 30));
stmt.registerOutParameter(1, Types.DOUBLE);
stmt.executeUpdate();
double result = stmt.getInt(1);
 ….
```

Indicate OUT parameters

Dealing with cursors

PL/SQL stored procedure

```
CREATE OR REPLACE PROCEDURE getDBUSERCursor(
                        p_username IN DBUSER.USERNAME%TYPE,
                        c_dbuser OUT SYS_REFCURSOR)
IS
BEGIN

  OPEN c_dbuser FOR
  SELECT * FROM DBUSER WHERE USERNAME LIKE p_username || '%';

END;
```

How to work with cursors using ResultSet

```
private static void callOracleStoredProcCURSORParameter()
                                        throws SQLException {
String getDBUSERCursorSql = "{call getDBUSERCursor(?,?)}";
try {
    CallableStatement call = conn.prepareCall(getDBUSERCursorSql);
    call.setString(1, "Juan");
    call.registerOutParameter(2, OracleTypes.CURSOR);
    call.executeUpdate();
    rs = (ResultSet) call.getObject(2);
    while (rs.next()) {
                String userid = rs.getString("USER_ID");
                String userName = rs.getString("USERNAME");
                String createdBy = rs.getString("CREATED_BY");
                System.out.println("UserName : " + userid);
                System.out.println("UserName : " + userName);
                System.out.println("CreatedBy : " + createdBy);
                }
    } catch (SQLException e) {
                System.out.println(e.getMessage());
    } finally {
                if (rs != null) rs.close();
                if (call != null) call.close();
                }}
```