Bases de Datos I
24/11/2020
Resumen Cap 10.3 24/11
IS-501
José Inestroza

10.3 Programación de bases de datos con llamadas a funciones y bibliotecas de clases: SQL / CLI y JDBC Embedded SQL

Se denomina enfoque de programación de bases de datos estáticas porque el texto de la consulta se escribe dentro del código fuente del programa y no se puede cambiar sin volver a compilar o reprocesar el código fuente. El uso de llamadas a funciones es un enfoque más dinámico para la programación de bases de datos que el SQL incorporado.

Una biblioteca de funciones, también conocida como interfaz de programación de aplicaciones (API), se utiliza para acceder a la base de datos.

La Interfaz de Nivel de Llamada SQL (SQL / CLI), que es parte del estándar SQL. Esto se desarrolló como una estandarización de la popular biblioteca de funciones conocida como ODBC (Open Database Connectivity). Usamos C como lenguaje anfitrión en nuestros ejemplos de SQL / CLI. Luego damos una descripción general de JDBC, que es la interfaz de función de llamada para acceder a bases de datos desde Java.

Comúnmente se asume que JDBC son las siglas de Java Database Connectivity, JDBC es solo una marca registrada de Sun Microsystems (ahora Oracle), no un acrónimo. La principal ventaja de utilizar una interfaz de llamada de función es que facilita el acceso a múltiples bases de datos dentro del mismo programa de aplicación, incluso si están almacenadas en diferentes paquetes DBMS.

10.3.1 Programación de base de datos con SQL / CLI usando C como lenguaje de host

Antes de usar las llamadas a funciones en SQL / CLI, es necesario instalar los paquetes de biblioteca apropiados en el servidor de la base de datos. Estos paquetes se obtienen del proveedor del DBMS que se está utilizando.

Maneja el entorno, la conexión, la declaración y la descripción registros. Cuando se utiliza SQL / CLI, las declaraciones SQL se crean dinámicamente y se pasan como parámetros de cadena en las llamadas a funciones. Por lo tanto, es necesario realizar un seguimiento de la información sobre las interacciones del programa host con la base de datos en las estructuras de datos en tiempo de ejecución porque los comandos de la base de datos se procesan en el tiempo de ejecución. La información se guarda en cuatro tipos de registros, representados como estructuras en tipos de datos C. Un registro de entorno se utiliza como contenedor para realizar un seguimiento de una o más conexiones de base de datos y para establecer la información del entorno. Un registro de conexión realiza

un seguimiento de la información necesaria para una conexión de base de datos en particular. Un registro de declaración realiza un seguimiento de la información necesaria para una declaración de SQL. Un registro de descripción realiza un seguimiento de la información sobre tuplas o parámetros, por ejemplo, el número de atributos y sus tipos en una tupla, o el número y tipos de parámetros en una llamada de función. Esto es necesario cuando el programador no conoce esta información sobre la consulta al escribir el programa. En nuestros ejemplos, asumimos que el programador conoce la consulta exacta, por lo que no mostramos ningún registro de descripción.

```
    //Program CLI1:
 0) #include sqlcli.h ;
 1) void printSal() {
 2) SQLHSTMT stmt1 ;
 3) SQLHDBC con1 ;
 4) SQLHENV env1 ;
 5) SQLRETURN ret1, ret2, ret3, ret4 ;
 6) ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1) ;
 7) if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit ;
 8) if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz",
      SQL_NTS) else exit ;
 9) if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit ;
10) SQLPrepare(stmt1, "select Lname, Salary from EMPLOYEE where Ssn = ?",
      SQL_NTS) ;
11) prompt("Enter a Social Security Number: ", ssn) ;
12) SQLBindParameter(stmt1, 1, SQL_CHAR, &ssn, 9, &fetchlen1) ;
13) ret1 = SQLExecute(stmt1) ;
14) if (!ret1) {
15)   SQLBindCol(stmt1, 1, SQL_CHAR, &lname, 15, &fetchlen1) ;
16)   SQLBindCol(stmt1, 2, SQL_FLOAT, &salary, 4, &fetchlen2) ;
17)   ret2 = SQLFetch(stmt1) ;
18)   if (!ret2) printf(ssn, lname, salary)
19)     else printf("Social Security Number does not exist: ", ssn) ;
20)   }
21) }
```

Pasos en un programa de base de datos. Al escribir un programa en C que incluirá llamadas a la base de datos a través de SQL / CLI, los siguientes son los pasos típicos que se toman. Ilustramos los pasos haciendo referencia al ejemplo CLI1 en la Figura 10.10, que lee el número de Seguro Social de un empleado e imprime el apellido y el salario del empleado.

1. Incluida la biblioteca de funciones. La biblioteca de funciones que comprende SQL / CLI debe estar incluida en el programa C. Esto se llama sqlcli.h y se incluye usando la línea 0 en la figura 10.10.

2. Declaración de variables de control. Declare las variables de control de tipos SQLHSTMT, SQLHDBC, SQLHENV y SQLHDESC para las declaraciones, conexiones, entornos y descripciones necesarias en el programa, respectivamente (líneas 2 a 4) .14 También declare variables de tipo SQLRETURN (línea 5) para contener la códigos de retorno de las

llamadas a funciones SQL / CLI. Un código de retorno de 0 (cero) indica la ejecución exitosa de la llamada a la función.

3. Registro ambiental. Se debe configurar un registro de entorno en el programa utilizando SQLAllocHandle. La función para hacer esto se muestra en la línea 6. Debido a que un registro de entorno no está contenido en ningún otro registro, el parámetro <handle_1> es el identificador NULL SQL_NULL_HANDLE (puntero NULL) al crear un entorno. El identificador (puntero) al registro de entorno recién creado se devuelve en la variable env1 en la línea 6.

4. Conexión a la base de datos. Se configura un registro de conexión en el programa utilizando SQLAllocHandle. En la línea 7, el registro de conexión creado tiene el identificador con1 y está contenido en el entorno env1. Luego se establece una conexión en con1 a una base de datos de servidor en particular utilizando SQLConnect.

5. **Registro de declaraciones.** Se configura un registro de declaración en el programa utilizando SQLAllocHandle. En la línea 9, el registro de sentencia creado tiene el identificador stmt1 y usa la conexión con1.
6. **Preparación de una declaración SQL y parámetros de la declaración**. La instrucción SQL se prepara utilizando la función SQL / CLI SQLPrepare. En la línea 10, esto asigna la cadena de instrucción SQL (la consulta en nuestro ejemplo) al identificador de instrucción stmt1. El símbolo del signo de interrogación (?) En la línea 10 representa un parámetro de declaración, que es un valor que se determinará en tiempo de ejecución, generalmente vinculándolo a una variable de programa en C. En general, puede haber varios parámetros en una cadena de instrucciones. Se distinguen por el orden de aparición de los signos de interrogación en la cadena de instrucciones (el primero? Representa el parámetro 1, el segundo? Representa el parámetro 2, y así sucesivamente). El último parámetro en SQLPrepare debe dar la longitud de la cadena de la instrucción SQL en bytes, pero si ingresamos la palabra clave SQL_NTS, esto indica que la cadena que contiene la consulta es una cadena terminada en NULL para que SQL pueda calcular la longitud de la cadena automáticamente. Este uso de SQL_NTS también se aplica a otros parámetros de cadena en las llamadas a funciones en nuestros ejemplos.
7. **Vinculación de los parámetros de la declaración**. Antes de ejecutar la consulta, todos los parámetros de la cadena de consulta deben estar vinculados a las variables del programa mediante la función SQLBindParameter de SQL / CLI. En la figura 10.10, el parámetro (indicado por?) De la consulta preparada referenciada por stmt1 está vinculado a la variable de programa C ssn en la línea 12. Si hay n parámetros en la declaración SQL, deberíamos tener n llamadas a la función SQLBindParameter, cada una con una posición de parámetro diferente (1, 2,…, n).

8. **Ejecución de la declaración. Después de estos preparativos**, ahora podemos ejecutar la instrucción SQL a la que hace referencia el identificador stmt1 usando la función SQLExecute (línea 13). Observe que, aunque la consulta se ejecutará en la línea 13, los resultados de la consulta aún no se han asignado a ninguna variable de programa C.
9. **Procesamiento del resultado de la consulta**. Para determinar dónde se devuelve el resultado de la consulta, una técnica común es el enfoque de columnas enlazadas. Aquí, cada columna en el resultado de una consulta está vinculada a una variable de programa en C utilizando la función SQLBindCol. Las columnas se distinguen por su orden de aparición en la consulta SQL. En la figura 10.10, líneas 15 y 16, las dos columnas de la consulta (Lname y Salario) están vinculadas a las variables del programa C lname y salario, respectivamente.

Figure 10.11
Program segment CLI2, a C program segment that uses SQL/CLI for a query with a collection of tuples in its result.

```
    //Program Segment CLI2:
 0) #include sqlcli.h ;
 1) void printDepartmentEmps() {
 2) SQLHSTMT stmt1 ;
 3) SQLHDBC con1 ;
 4) SQLHENV env1 ;
 5) SQLRETURN ret1, ret2, ret3, ret4 ;
 6) ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1) ;
 7) if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit ;
 8) if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz",
      SQL_NTS) else exit ;
 9) if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit ;
10) SQLPrepare(stmt1, "select Lname, Salary from EMPLOYEE where Dno = ?",
      SQL_NTS) ;
11) prompt("Enter the Department Number: ", dno) ;
12) SQLBindParameter(stmt1, 1, SQL_INTEGER, &dno, 4, &fetchlen1) ;
13) ret1 = SQLExecute(stmt1) ;
14) if (!ret1) {
15)   SQLBindCol(stmt1, 1, SQL_CHAR, &lname, 15, &fetchlen1) ;
16)   SQLBindCol(stmt1, 2, SQL_FLOAT, &salary, 4, &fetchlen2) ;
17)   ret2 = SQLFetch(stmt1) ;
18)   while (!ret2) {
19)     printf(lname, salary) ;
20)     ret2 = SQLFetch(stmt1) ;
21)     }
22)   }
```

10. **Recuperando valores de columna.** Finalmente, para recuperar los valores de la columna en las variables del programa C, se usa la función SQLFetch (línea 17). Esta función es similar al comando FETCH del SQL incorporado. Si el resultado de una consulta tiene una colección de tuplas, cada llamada SQLFetch obtiene la siguiente tupla y devuelve sus valores de columna en las variables de programa enlazadas. SQLFetch devuelve un código de excepción (distinto de cero) si no hay más tuplas en el resultado de la consulta. Como podemos ver, el uso de llamadas a funciones dinámicas requiere mucha preparación para configurar las declaraciones SQL y vincular los parámetros de las declaraciones y los resultados de las consultas a las variables de programa adecuadas. En CLI1, la consulta SQL selecciona una única tupla. La figura 10.11 muestra un ejemplo de recuperación de múltiples tuplas. Suponemos que las variables apropiadas del programa C se han declarado como en la Figura 10.1. El segmento de programa en CLI2 lee (entradas) un número de departamento y luego

recupera los empleados que trabajan en ese departamento. Luego, un bucle recorre cada registro de empleado, uno a la vez, e imprime el apellido y el salario del empleado.

Controladores JDBC. Un controlador JDBC es básicamente una implementación de las clases y los objetos asociados y las llamadas a funciones especificadas en JDBC para el RDBMS de un proveedor en particular. Por lo tanto, un programa Java con objetos JDBC y llamadas a funciones puede acceder a cualquier RDBMS que tenga un controlador JDBC disponible. Dado que Java está orientado a objetos, sus bibliotecas de funciones se implementan como clases.
Antes de poder procesar llamadas a funciones JDBC con Java, es necesario importar las bibliotecas de clases JDBC, que se denominan java.sql. *. Estos se pueden descargar e instalar a través de la Web.19 JDBC está diseñado para permitir que un solo programa Java se conecte a varias bases de datos diferentes. A veces se denominan fuentes de datos a las que accede el programa Java y pueden almacenarse utilizando RDBMS de diferentes proveedores que residen en diferentes máquinas. Por lo tanto, diferentes accesos a fuentes de datos dentro del mismo programa Java pueden requerir controladores JDBC de diferentes proveedores. Para lograr esta flexibilidad, se emplea una clase JDBC especial llamada clase de administrador de controladores, que realiza un seguimiento de los controladores instalados. Un controlador debe registrarse con el administrador de controladores antes de que se utilice. Las operaciones (métodos) de la clase de administrador de controladores incluyen getDriver, registerDriver y deregisterDriver. Estos se pueden usar para agregar y eliminar controladores para diferentes sistemas de forma dinámica. Otras funciones configuran y cierran conexiones a fuentes de datos. Para cargar un controlador JDBC explícitamente, se puede utilizar la función genérica de Java para cargar una clase. Por ejemplo, para cargar el controlador JDBC para Oracle RDBMS, se puede utilizar el siguiente comando:
Class.forName ("oracle.jdbc.driver.OracleDriver")

Esto registrará el controlador con el administrador de controladores y lo pondrá a disposición del programa. También es posible cargar y registrar los controladores necesarios en la línea de comando que ejecuta el programa, por ejemplo, incluyendo lo siguiente en el línea de comando:
-Djdbc.drivers = oracle.jdbc.driver
Pasos de programación de JDBC. Los siguientes son pasos típicos que se toman cuando escribir un programa de aplicación Java con acceso a la base de datos a través de llamadas a funciones JDBC. Ilustramos los pasos haciendo referencia al ejemplo JDBC1 en la Figura 10.12, que lee el número de Seguro Social de un empleado e imprime el apellido y el salario del empleado.

```
   //Program JDBC1:
 0) import java.io.* ;
 1) import java.sql.*
    ...
 2) class getEmpInfo {
 3)   public static void main (String args []) throws SQLException, IOException {
 4)     try { Class.forName("oracle.jdbc.driver.OracleDriver")
 5)     } catch (ClassNotFoundException x) {
 6)       System.out.println ("Driver could not be loaded") ;
 7)     }
 8)     String dbacct, passwrd, ssn, lname ;
 9)     Double salary ;
10)     dbacct = readentry("Enter database account:") ;
11)     passwrd = readentry("Enter password:") ;
12)     Connection conn = DriverManager.getConnection
13)       ("jdbc:oracle:oci8:" + dbacct + "/" + passwrd) ;
14)     String stmt1 = "select Lname, Salary from EMPLOYEE where Ssn = ?" ;
15)     PreparedStatement p = conn.prepareStatement(stmt1) ;
16)     ssn = readentry("Enter a Social Security Number: ") ;
17)     p.clearParameters() ;
18)     p.setString(1, ssn) ;
19)     ResultSet r = p.executeQuery() ;
20)     while (r.next()) {
21)       lname = r.getString(1) ;
22)       salary = r.getDouble(2) ;
23)       system.out.printline(lname + salary) ;
24)   } }
25) }
```

Figure 10.12
Program segment JDBC1, a Java program segment with JDBC.

1. **Importe la biblioteca de clases JDBC**. La biblioteca de clases JDBC debe importarse al programa Java. Estas clases se denominan java.sql. * Y se pueden importar utilizando la línea 1 de la figura 10.12. También se debe importar cualquier biblioteca de clases Java adicional que necesite el programa.

2. **Cargue el controlador JDBC**. Esto se muestra en las líneas 4 a 7. La excepción de Java en la línea 5 ocurre si el controlador no se carga correctamente.
3. **Cree variables apropiadas**. Estas son las variables necesarias en el programa Java (líneas 8 y 9).
4. **El objeto Conexión**. Un objeto de conexión se crea utilizando el Función getConnection de la clase DriverManager de JDBC. En las líneas 12 y 13, el objeto Conexión se crea mediante la llamada de función.getConnection (urlstring), donde urlstring tiene la forma jdbc: oracle: <driverType>: <dbaccount> / <password> Una forma alternativa es getConnection (url, dbaccount, password) Se pueden establecer varias propiedades para un

objeto de conexión, pero son principalmente relacionado con propiedades transaccionales, que discutimos en el Capítulo 21.

5. **El objeto Declaración preparada.** Se crea un objeto de declaración en el programa. En JDBC, hay una clase de instrucción básica, Statement, con dos subclases especializadas: PreparedStatement y CallableStatement. El ejemplo de la figura 10.12 ilustra cómo se crean y utilizan los objetos PreparedStatement. El siguiente ejemplo (Figura 10.13) ilustra el otro tipo de objetos Statement. En la línea 14 de la Figura 10.12, una cadena de consulta con un solo parámetro, indicado por? símbolo: se crea en la variable de cadena stmt1. En la línea 15, se crea un objeto p de tipo PreparedStatement basado en la cadena de consulta en stmt1 y usando el objeto de conexión conn. En general, el programador debería usar objetos PreparedStatement si una consulta se va a ejecutar varias veces, ya que se prepararía, comprobaría y compilaría solo una vez, ahorrando así este costo para las ejecuciones adicionales de la consulta.
6. **Configuración de los parámetros de la declaración.** El símbolo del signo de interrogación (?) En la línea 14 representa un parámetro de declaración, que es un valor que se determina en tiempo de ejecución, normalmente vinculándolo a una variable de programa Java. En general, podría haber varios parámetros, que se distinguen por el orden de aparición de los signos de interrogación dentro de la cadena de instrucciones (el primero? Representa el parámetro 1, el segundo? Representa el parámetro 2, y así sucesivamente), como hemos comentado anteriormente.
7. **Vinculación de los parámetros de la declaración.** Antes de ejecutar una consulta PreparedStatement, cualquier parámetro debe estar vinculado a las variables del programa. Dependiendo del tipo de parámetro, diferentes funciones como setString, setInteger, setDouble, etc. se aplican al objeto PreparedStatement para establecer sus parámetros. Se debe utilizar la función adecuada para que corresponda con el tipo de datos del parámetro que se está configurando. En la figura 10.12, el parámetro (indicado por?) En el objeto p está vinculado a la variable de programa Java ssn en la línea 18. La función setString se usa porque ssn es una variable de cadena. Si hay n parámetros en la declaración SQL, deberíamos tener n funciones set ..., cada una con una posición de parámetro diferente (1, 2, ..., n). Por lo general, se recomienda borrar todos los parámetros antes de establecer nuevos valores (línea 17).

```
   //Program Segment JDBC2:
0) import java.io.* ;
1) import java.sql.*
   ...
2) class printDepartmentEmps {
3)   public static void main (String args [])
         throws SQLException, IOException {
4)     try { Class.forName("oracle.jdbc.driver.OracleDriver")
5)     } catch (ClassNotFoundException x) {
6)       System.out.println ("Driver could not be loaded") ;
7)     }
8)     String dbacct, passwrd, lname ;
9)     Double salary ;
10)    Integer dno ;
11)    dbacct = readentry("Enter database account:") ;
12)    passwrd = readentry("Enter password:") ;
13)    Connection conn = DriverManager.getConnection
14)      ("jdbc:oracle:oci8:" + dbacct + "/" + passwrd) ;
15)    dno = readentry("Enter a Department Number: ") ;
16)    String q = "select Lname, Salary from EMPLOYEE where Dno = " +
       dno.tostring() ;
17)    Statement s = conn.createStatement() ;
18)    ResultSet r = s.executeQuery(q) ;
19)    while (r.next()) {
20)      lname = r.getString(1) ;
21)      salary = r.getDouble(2) ;
22)      system.out.printline(lname + salary) ;
23)  } }
24) }
```

Figure 10.13
Program segment JDBC2, a Java program segment that uses JDBC for a query with a collection of tuples in its result.

8. Ejecución de la instrucción SQL. Después de estos preparativos, ahora podemos ejecutar la instrucción SQL a la que hace referencia el objeto p usando la función executeQuery (línea 19). Hay una función genérica que se ejecuta en JDBC, más dos funciones especializadas: executeUpdate y executeQuery. executeUpdate se utiliza para las sentencias de inserción, eliminación o actualización de SQL, y devuelve un valor entero que indica el número de tuplas afectadas. executeQuery se usa para declaraciones de recuperación de SQL y devuelve un objeto de tipo ResultSet, que discutiremos a continuación.

9. Procesando el objeto ResultSet. En la línea 19, el resultado de la consulta es devuelto en un objeto r de tipo ResultSet. Esto se asemeja a una matriz bidimensional o una tabla, donde las tuplas son las filas y los atributos devueltos son las columnas. Un objeto ResultSet es similar a un cursor en SQL incorporado y un iterador en SQLJ. En nuestro ejemplo, cuando se ejecuta la consulta, r se refiere a una tupla antes de la primera tupla en el resultado de la consulta. La función r.next () (línea 20) se mueve a la siguiente tupla (fila) en el objeto ResultSet y devuelve NULL si no hay más objetos. Esto se utiliza para controlar el bucle. El programador puede

referirse a los atributos en la tupla actual usando varios get ... funciones que dependen del tipo de cada atributo (por ejemplo, getString, getInteger, getDouble, etc.). El programador puede utilizar las posiciones de los atributos (1, 2) o los nombres de los atributos reales ("Lname", "Salario") con las funciones get…. En nuestros ejemplos, usamos la notación posicional en las líneas 21 y 22.

En general, el programador puede verificar las excepciones de SQL después de cada llamada a la función JDBC. No hicimos esto para simplificar los ejemplos. Tenga en cuenta que JDBC no distingue entre consultas que devuelven tuplas únicas y aquellas que devuelven tuplas múltiples, a diferencia de algunas de las otras técnicas. Esto se justifica porque un solo conjunto de resultados de tupla es solo un caso especial. En el ejemplo JDBC1, la consulta SQL selecciona una sola tupla, por lo que el ciclo en las líneas 20 a 24 se ejecuta como máximo una vez. El ejemplo que se muestra en la Figura 10.13 ilustra la recuperación de múltiples tuplas. El segmento de programa en JDBC2 lee (ingresa) un número de departamento y luego recupera los empleados que trabajan en ese departamento. Luego, un bucle recorre cada registro de empleado, uno a la vez, e imprime el apellido y el salario del empleado. Este ejemplo también ilustra cómo podemos ejecutar una consulta directamente, sin tener que prepararla como en el ejemplo anterior. Esta técnica se prefiere para consultas que se ejecutarán una sola vez, ya que es más simple de programar.
En la línea 17 de la Figura 10.13, el programador crea un objeto Statement (en lugar de PreparedStatement, como en el ejemplo anterior) sin asociarlo con una cadena de consulta en particular. La cadena de consulta q se pasa al objeto de instrucción s cuando se ejecuta en la línea 18.
Con esto concluye nuestra breve introducción a JDBC. Se remite al lector interesado al sitio web http://java.sun.com/docs/books/tutorial/jdbc/, que contiene muchos más detalles sobre JDBC.