

Teórico 5

APIs para Acceso a Bases de Datos



Arquitecturas de Aplicaciones con Accesos a Base de Datos

Básicamente tenemos dos Arquitecturas:

- Dos Capas

- La aplicación cliente se conecta al servidor de DB directamente.

- Tres Capas

- La aplicación cliente se conecta al servidor de DB a través de un servidor de aplicaciones.

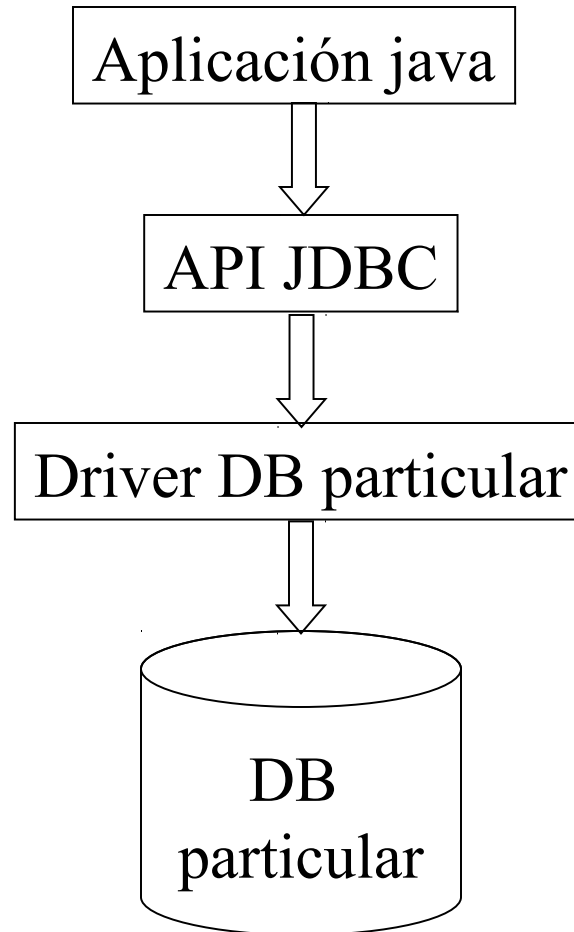
Java DataBase Connectivity (JDBC)

Es una API JAVA que permite la ejecución de operaciones sobre base de datos, independientemente de la base de datos a la cual se accede, utilizando SQL de la base de datos base de datos que se utilice.

Principales características de JDBC:

- Interacción con la DB a través de SQL.
- 100% Java.
- Muy simple de utilizar.
- Alta performance.
- Se puede utilizar cualquier base de datos para la cual este implementado el driver de acceso, por ej.: MySQL, Firebird, Postgres, Oracle, etc.

Arquitectura JDBC



Algunos Componentes de la API

- Clase estática DriverManager
 - Carga el driver de la base de datos elegida.
- Interfaz Connection
 - Representa una conexión a la base de datos.
- Interfaz Statement y subclases
 - Representa una sentencia SQL.
- Interfaz ResultSet
 - Para tratar registros retornados por la ejecución de un Statement.
- Interfaz DatabaseMetaData
 - Posee métodos para extraer información sobre los metadatos, por ejemplo, tablas , atributos de las tablas, etc.

URL JDBC

Las URL JDBC se componen de la siguiente forma:

`jdbc:subprotocolo:fuentes`

- Cada Driver tiene su propio subprotocolo.
- Cada subprotocolo tiene su propia sintaxis para la fuente.

Ejemplos de URLs JDBC

- PostgreSQL

`jdbc:postgresql://host[:port]/database`

Ejemplo:

`jdbc:postgresql://localhost:5432/prueba`

- Mysql

`jdbc:mysql://host[:port]/database`

Ejemplo:

`jdbc:mysql://localhost:3306/ejercicio1`

Clase DriverManager

- Provee el método

`getConnection(String url, String user, String password)`

Este método retorna un objeto que implementa la Interfaz Connection que representa una sesión de conexión con la Base de Datos.

Interfaz Connection

- Representa una sesión con una base de datos particular.
- En la sesión se ejecutan instrucciones SQL y se obtienen los resultados.
- Se puede obtener información de “metadata”(estructura) de la base de datos.
- Provee métodos para manejo de transacciones (commit(), rollback())

Interfaz Connection (cont.)

La Interfaz provee métodos para la creación de objetos que implementan las interfaces Statement, PreparedStatement y CallableStatement, que representan una sentencia SQL a ejecutar en el motor de bases de datos.

Interfaz Connection (cont.)

`setAutoCommit(boolean)`

- Si es Verdadero, cada ejecución de una sentencia SQL se hace es una transacción.
- Si AutoCommit es Falso, se debe utilizar explícitamente los métodos `commit` o `rollback` para delimitar la transacción.

Ejemplo para obtener una conexión a mysql

```
String driver = "org.gjt.mm.mysql.Driver";  
String url = "jdbc:mysql://localhost/prueba";  
String username = "root";  
String password = "root";  
// Load database driver if not already loaded.  
Class.forName(driver);  
// Establish network connection to database.  
Connection connection =  
DriverManager.getConnection(url, username, password);
```

Interfaz Statement

- Representa una sentencia SQL estática, es decir, no se le puede pasar parámetros.

Interfaz PreparedStatement

- Hereda de Statement y representa un SQL precompilado, permite el pasaje de parámetros.

Interfaz CallableStatement

- Hereda de Statement, su objetivo es la ejecución de procedimientos almacenados de la base de datos.

Métodos de la Interfaz Connection para la Creación de Statements

La Interfaz Connection provee métodos para crear diferentes tipos de Statement:

Statement createStatement()

- Retorna un nuevo objeto Statement.

PreparedStatement prepareStatement(String sql)

- Retorna un nuevo objeto PreparedStatement.

CallableStatement prepareCall(String sql)

- Retorna un nuevo objeto CallableStatement.

- La optimización es el objetivo de tener diferentes tipos de Statement.

Parámetros para la Creación de Statement

- A la creación de Statement se le puede pasar varios parámetros.
- Un parámetro es el tipo de ResultSet a retornar, puede tomar los siguientes valores:
 - `ResultSet.TYPE_FORWARD_ONLY`: Sólo avanza hacia adelante.
 - `ResultSet.TYPE_SCROLL_INSENSITIVE`: puede ser recorrido en ambos sentidos y las actualizaciones no se reflejan hasta que se haga la consulta de nuevo.
 - `ResultSet.TYPE_SCROLL_SENSITIVE`. puede ser recorrido en ambos sentidos y las actualizaciones se reflejan cuando se producen.

Parámetros en la creación de Statement (Cont.)

- Otro parámetro es para determinar si el ResultSet es actualizable o no.
- Puede tomar los siguientes valores:
 - `ResultSet.CONCUR_READ_ONLY`: El resultado es de sólo lectura.
 - `ResultSet.CONCUR_UPDATABLE`.: El resultado es Actualizable.

Algunos Métodos de Statement

`ResultSet executeQuery(String)`

- Ejecuta una sentencia SQL que retorna un `ResultSet`.

`int executeUpdate(String)`

- Ejecuta sentencia SQL del tipo INSERT, UPDATE o DELETE. Retorna el numero de registros afectados por la ejecución de la sentencia.

`boolean execute(String)`

- Ejecuta un sentencia SQL que puede retornar múltiples resultados.

Interfaz ResultSet

- Un ResultSet provee acceso a una tabla de datos generada por la ejecución de una sentencia SQL, a través de un Statement.
- Un único ResultSet por Statement puede ser abierto al mismo tiempo.
- Las filas de la tabla se pueden recuperar en secuencia o no, dependiendo de como se creó el Statement que retorna el ResultSet.
- Un ResultSet mantiene un cursor que apunta a la fila o registro actual de la tabla.

Métodos de ResultSet

- **boolean next():** Corre el cursor a la próxima fila. La primera llamada next() activa la primer fila. Retorna falso cuando no quedan filas.
- **void close():** Libera el ResultSet.
- **<Type> get<Type>(int columnIndex):** Retorna el valor(del tipo *Type*) del campo para la fila actual. ColumnIndex comienza de 1. Ejemplo: getString(1), retorna el valor String de la columna 1 del ResultSet.
- **<Type> get<Type>(String columnName):** Ídem anterior pero tiene como parámetro el nombre de la columna. Es menos eficiente que el anterior.
- **update<Type>(String columnLabel, <Type> valor):** Actualiza lo almacenado en la fila actual en la columna *columnLabel* con el valor *valor*.
- **int findColumn(String columnName):** Retorna el índice de la columna con nombre columnName.



Métodos de ResultSet (Cont.)

- **previous():** mueve el cursor una fila atrás. Retorna true si el cursor se pudo posicionar, sino false.
- **first():** mueve el cursor a la primera fila. Retorna true si el cursor se posicionó en la primera fila, retorna false si el ResultSet no tiene filas.
- **last():** mueve el cursor a la última fila. Retorna true si el cursor se posicionó en la última fila, retorna false si el ResultSet es vacío.
- **beforeFirst():** posiciona el cursor al comienzo del ResultSet, antes de la primera fila. Si el ResultSet es vacío el método no tiene efecto.
- **afterLast():** posiciona el cursor al final del ResultSet, después de la última fila. Si el ResultSet es vacío el método no tiene efecto.
- **relative(int cant):** mueve el cursor *cant* filas relativas a la posición corriente.
- **absolute(int n):** posiciona el cursor a la fila *n* del ResultSet.

Ejemplo

```
Statement statement = connection.createStatement();
```

```
String query = "SELECT * FROM persona";
```

```
//Envía el query a la base de datos y almacena el resultado
```

```
ResultSet resultSet = statement.executeQuery(query);
```

```
// Muestra los resultados.
```

```
while(resultSet.next())
```

```
{
```

```
    System.out.print(" DNI: " + resultSet.getString("DNI"));
```

```
    System.out.print("; Nombre: " + resultSet.getString("nombre"));
```

```
    System.out.print("; Email: " + resultSet.getString("email"));
```

```
    System.out.print("\n    ");
```

```
    System.out.print("\n    ");
```

```
}
```

Extracción de Metadatos

- Para obtener metadatos de las bases de datos se utiliza la Interfaz DatabaseMetaData, por medio de esta Interfaz se puede obtener por ejemplo la lista de tablas o las columnas de una tabla.

Algunos Métodos de la Interfaz

ResultSet **getTables**(String catalog, String schemaPattern, String tableNamePattern, String[] types)

Recupera una descripción de las tablas disponibles en un catálogo.
Retorna un resultSet de registros con los siguientes campos:

TABLE_CAT: String => catalogo de la tabla(puede ser null)

TABLE_SCHEM: String => Esquema de la tabla (puede ser null)

TABLE_NAME: String => nombre de la tabla.

TABLE_TYPE: String => tipo de la tabla. Tipos típicos son "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS", "SYNONYM".

REMARKS: String => comentarios en la tabla

Las listadas debajo no están implementadas en la mayoría de los driver, por ej. MySQL y Postgres

TYPE_CAT: String => Los tipos del catalogo (puede ser null)

TYPE_SCHEM: String => los tipos del esquema (puede ser null)

TYPE_NAME: String => nombre del tipo (puede ser null)

SELF_REFERENCING_COL_NAME: String => nombre de la columna “identificador” de la tabla tipada (puede ser null)

REF_GENERATION: String => Especifica como los valores en SELF_REFERENCING_COL_NAME son creados. Los valores son "SYSTEM", "USER", "DERIVED". (puede ser null)



Algunos Métodos de la Interfaz (cont.)

- `ResultSet getColumn(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)`

Recupera una descripción de las columnas de una tabla en un catálogo específico

Los tipos de datos de la columnas son definidos en `java.sql.Types`

•

Algunos Métodos de la Interfaz (cont.)

ResultSet **getProcedures**(**String** catalog, **String** schemaPattern, **String** procedureNamePattern)

Recupera una descripción de los procedimientos almacenados disponibles.

Los tipos de parámetros están dados en DatabaseMetaData con las constantes `procedureColumnIn` , `procedureColumnOut`, etc.

Los tipos de datos de los parámetros son definidos en `java.sql.Types`

Ejemplo de Extracción de Metadatos de una Base de Datos Oracle

El ejemplo muestra los esquemas de una base de datos Oracle

```
String driver = "oracle.jdbc.driver.OracleDriver";
String url = "jdbc:oracle:thin:@//localhost:1521/XE";
String username = "USUARIO1";
String password = "USUARIO1";
// Carga el driver de la base de datos
    Class.forName(driver);

// Establece la conexión con la base de datos.
Connection connection = DriverManager.getConnection(url, username, password);
DatabaseMetaData metaData = connection.getMetaData();
ResultSet resultSetSchemas = metaData.getSchemas();
System.out.println(" Esquemas de la Base de datos ");
while(resultSetSchemas.next()) {
    System.out.println(" esquema: " + resultSetSchemas.getString(1));
}
```



Ejemplo de Extracción de Metadatos de Tablas en PostgreSQL

```
String driver = "org.postgresql.Driver";
String url = "jdbc:postgresql://localhost:5432";
String username = "postgres";
String password = "root";
Class.forName(driver);
Connection connection = DriverManager.getConnection(url, username, password);
String[] tipo = {"TABLE"};
DatabaseMetaData metaData = connection.getMetaData();
ResultSet resultSetTables = metaData.getTables("Proyecto", "comparador", null, tipo);
System.out.println(" tablas de la base de datos ");
while(resultSetTables.next()) {
    System.out.println(" catalogo: " + resultSetTables.getString(1));
    System.out.println(" esquema: " + resultSetTables.getString(2));
    System.out.println(" nombre: " + resultSetTables.getString(3));
    System.out.println(" tipo: " + resultSetTables.getString(4));
    System.out.println(" comentarios: " + resultSetTables.getString(5));
}
```



Acceso al Catalogo

- Hay alguna información de metadatos que no pueden ser extraídas por medio de los métodos que provee la Interfaz Metadata, esta información puede ser recuperada directamente del catalogo del motor de bases de datos.

Ejemplo de Acceso al Catalogo de MySQL

- Mysql guarda información de metadatos en las base de datos `information_schema` y `mysql`, en tanto PostgreSQL en `c/base de datos` existen dos catalogos, ANSI y PostgreSQL.
- El siguiente ejemplo extrae del catalogo el nombre y código de los procedimientos del esquema “procedimientos”

```
SELECT
```

```
specific_name AS nombre, routine_definition AS codigo
```

```
FROM information_schema.ROUTINES R
```

```
WHERE routine_schema = 'procedimientos'
```



Correspondencia de tipos SQL y JAVA

Tipos SQL

CHAR, VARCHAR, LONGVARCHAR
NUMERIC, DECIMAL
BIT
TINYINT
SMALLINT
INTEGER
BIGINT
REAL
FLOAT, DOUBLE
BINARY, VARBINARY, LONGVARBINARY
DATE
TIME
TIMESTAMP

Tipos Java

String
java.math.BigDecimal
boolean
byte
short
int
long
float
double
byte[]
java.sql.Date
java.sql.Time
java.sql.Timestamp



Persistencia de Objetos (en Java)

- Existen varios estándares para la persistencia de objetos en Java.
- Definen los mecanismos para hacer persistentes objetos java en base de datos relacionales u objeto relacionales.

Estándares

- Para la persistencia de objetos en java existen varios estándares:
 - JDO (desde 2001), implementaciones:
 - TJDO(JDO 1).
 - JPOX (JDO 1, 2.0, 2.1).
 - DataNucleus (JDO 1, 2.0, 2.1, 2.2, 2.3).
 - Kodo (JDO 1, 2.0).
 - JPA (desde 2006, forma parte de EJB 3.0), implementaciones:
 - JPOX.
 - OpenJPA.
 - DataNucleus.
 - Hibernate.



Esquema General

