Teórico 7

APIs para Acceso a Bases de Datos

Arquitecturas de Aplicaciones con Accesos a Base de Datos

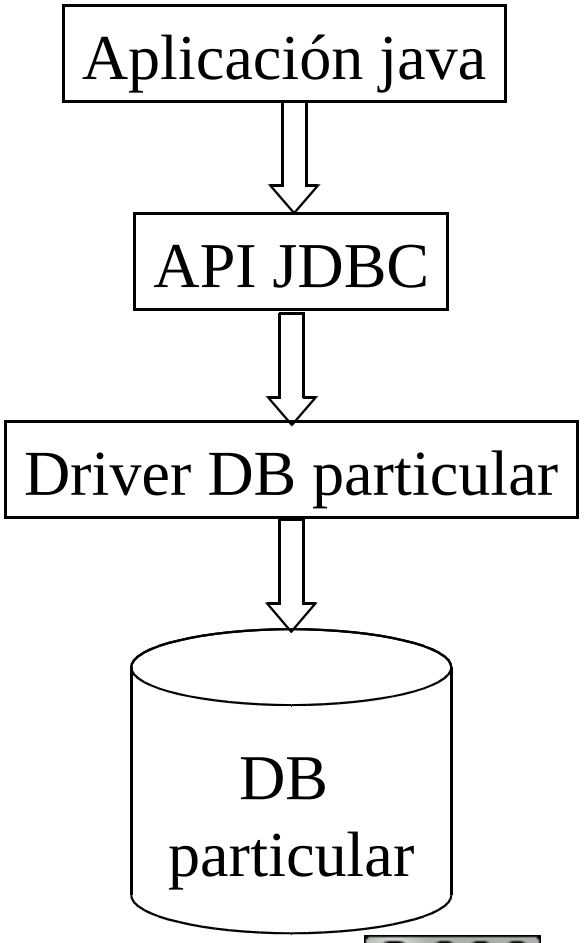
Básicamente tenemos dos Arquitecturas:

- 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.
- Su última especificación (versión 4.0) se puede descargar de http://java.sun.com/products/jdbc/download.html
- Principales características de JDBC:
  - ✓ Interacción con la DB a través de SQL.
  - ✓ 100% Java.
  - ✓ Muy simple de utilizar.
  - ✓ Alta performance.
  - ✓ Se puede utilizar en cualquier motor de bases de datos para la cual este implementado el driver de acceso, por ej.: MySQL, Firebird, PostgreSQL, Oracle, etc.

Arquitectura JDBC



Componentes de la API

- Clase estática DriverManager
  - Carga el driver de la base de datos elegida
- Interface Connection
  - Representa una conexión a la base de datos.
- Interface Statement y subclases
  - Representa una sentencia SQL.
- Interface ResultSet
  - Para tratar registros retornados por la ejecución de un Statement.
- 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:fuente
```

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

Ejemplos de URLs JDBC

- Firebird

  `jdbc:firebirdsql:`*`//host[/port]:database`*

  Ejemplo:

  ```
  jdbc:firebirdsql:localhost/3050:D:/BASEDATO/
  prueba.fdb";
  ```

- 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 conexión que representa la conexión con la DB.

Interfase 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())

Algunos Métodos de la Interfase

`Statement createStatement()`

- Retorna un nuevo objeto statement.

`PreparedStatement prepareStatement(String sql)`

- Retorna un nuevo objeto PreparedStatement.

`CallableStatement prepareCall(String sql)`

- Retorno a nuevo objeto CallableStatement.

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

Método setAutoCommit

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 MySQLl

```
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);
```

Interfase Statement

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

Interfase PreparedStatement

- Representa un SQL precompilado, permite el pasaje de parámetros.

Interfase CallableStatement

- Su objetivo es la ejecución de procedimientos almacenados de la base de datos.

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 número de registros afectados por la ejecución de la sentencia.

`boolean execute(String)`

- Ejecuta un sentencia SQL que puede retornar múltiples resultados.

ResultSet

- Un ResultSet provee acceso a una tabla de datos generada por la ejecución de un sentencia SQL, a través de un Statement.
- Un único ResultSet por Statement puede ser abierto al mismo tiempo.
- Las filas de la tabla se recuperan en secuencia.
- Un ResultSet mantiene un cursor que apunta a la fila o registro actual de la tabla.
- El método next() mueve el cursor a la siguiente fila.
- No es posible hacer un rewind del ResultSet.

Metodos de ResultSet

- boolean next()
  - Corre el cursor a la próxima fila.
  - La primera llamada next() activa la primer fila.
  - Retorna falso cuando no hay mas 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 ResulSet.
- <*Type*> get<*Type*>(String columnName)
  - Ídem anterior pero tiene como parámetro el nombre de la columna
  - Es menos eficiente que el anterior.
- int findColumn(String columnName)
  - Retorna el índice de la columna con nombre columnName.

Ejemplo

```
 Statement statement = connection.createStatement();
 String query = "SELECT * FROM persona";

//Envía en query a la base de datos y almacena el resulatdo.
 ResultSet resultSet = statement.executeQuery(query);

// Muestra los resultados.
 while(resultSet.next())
 {
  System.out.print(" DNI: " + resultSet.getString(1));
  System.out.print("; Nombre: " + resultSet.getString(2));
  System.out.print("; Email: " + resultSet.getString(3)) ;
  System.out.print("\n   ");
  System.out.print("\n   ");
 }
```

Ejemplo: Extracción de Metadatos de una Base de Datos Mysql

```
String driver = "org.gjt.mm.mysql.Driver";
String url = "jdbc:mysql://localhost/prueba";
String username = "root";
String password = "root";

// Carga el driver de la base de datos
     Class.forName(driver);

// Establish network connection to database.
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("  " + resultSetSchemas.getString(1));
    }
```

Correspondencia de tipos SQL y JAVA

Tipos SQL	**Tipos Java**
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	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.
    - Hiberanate.

Esquema General