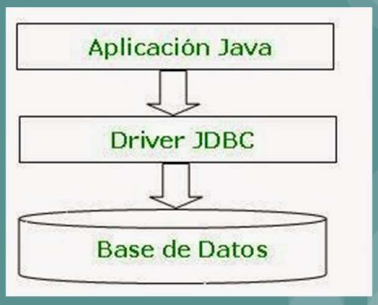
Bases de datos (JDBC)

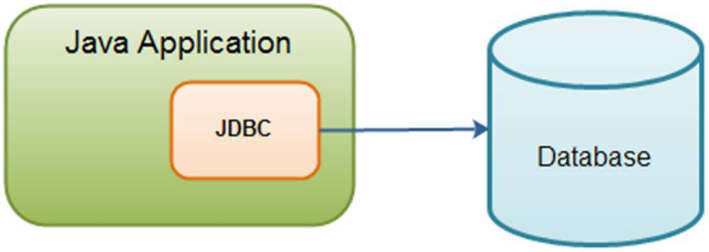
Módulo de Programación.
CFGS Desarrollo de Aplicaciones Multiplataforma
IES Segundo de Chomón



Java JDBC

La API Java JDBC (Java Database Connectivity) permite que las aplicaciones Java se conecten a bases de datos relacionales como MySQL, PostgreSQL, MS SQL Server, Oracle, etc para hacer consultas, actualizaciones, llamadas a procedimientos almacenados y obtener metadatos sobre la base de datos.

La API Java JDBC es parte del SDK básico de Java SE.

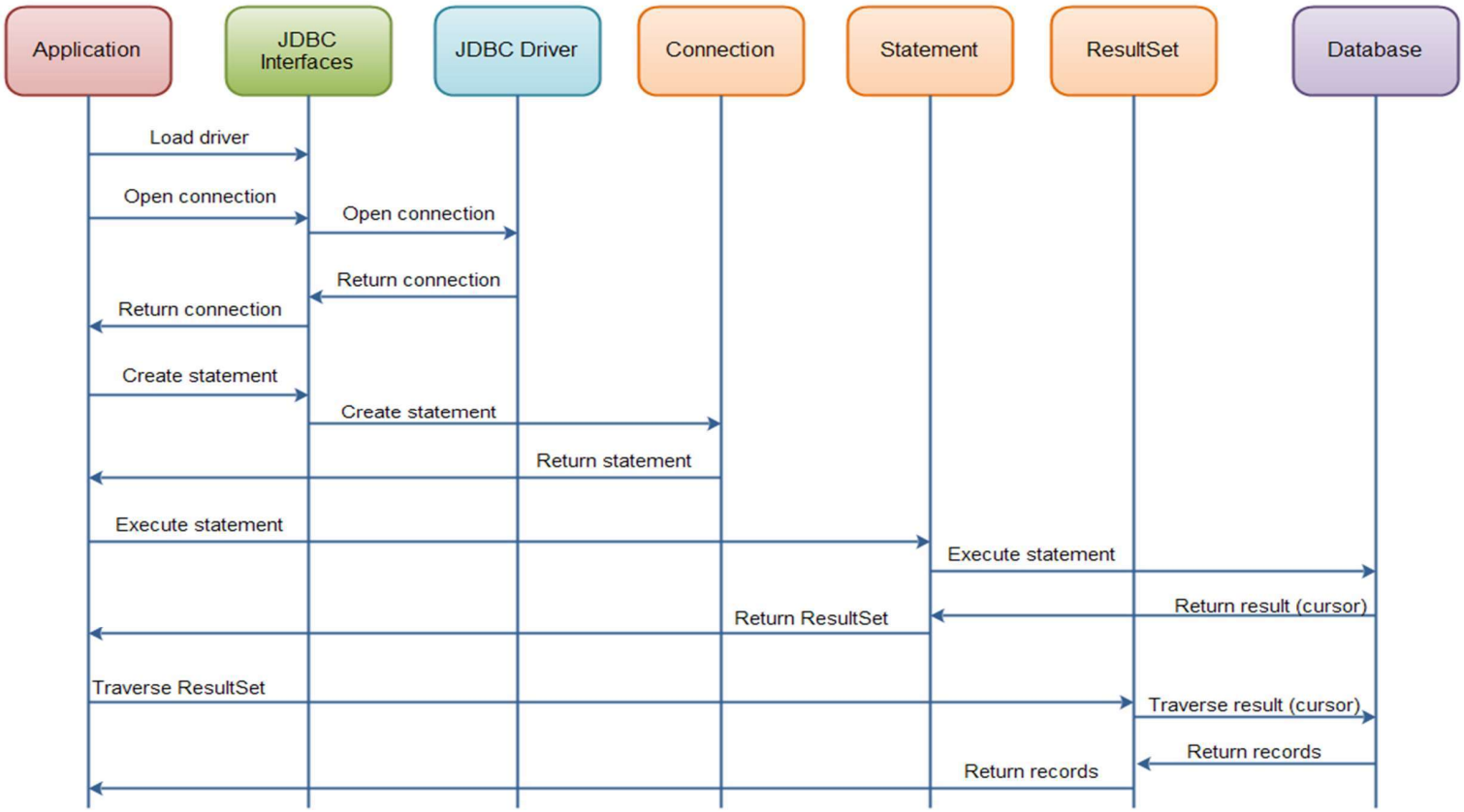


Java JDBC

- Es independiente de la base de datos.
- No es independiente de SQL.
- No es para bases de datos no relacionales.
- Pasos típicos de una aplicación:
  - Cargar controlador JDBC (opcional desde Java 6).
  - Abrir una conexión con la base de datos.
  - Crear un objeto `Statement`.
  - Actualizar la base de datos con una sentencia SQL.
  - Consultar la base de datos con una sentencia SQL.
  - Cerrar las conexiones.

Ejemplo

Más información



JDBC Driver

Un driver JDBC es un conjunto de clases Java que implementan las interfaces JDBC y se dirigen a una base de datos específica.

Los driver JDBC generalmente los proporciona el proveedor de la base de datos, pero a veces los puede proporcionar la comunidad de desarrolladores en torno a una base de datos.

Más información

JDBC Connection

La clase `Connection` representa una conexión de base de datos a una base de datos relacional.

Esta conexión puede ser de tres formas:

- Con URL.
- Con URL, usuario y contraseña.
- Con URL y propiedades.

Más información

Se recomienda usar `try-with-resources` para cerrar automáticamente la conexión.

```java
try (Connection connection = DriverManager.getConnection(url);
    Statement statement = connection.createStatement();
    ResultSet resultset = statement.executeQuery("SELECT * FROM contactos")) {

  while (resultset.next()) {
    String name = resultset.getString("nombre");
    int phone = resultset.getInt("telefono");
    String email = resultset.getString("email");
    System.out.println(name + "\t" + phone + "\t" + email);
  }

} catch (SQLException e) {
  System.out.println("Error en la conexión de la base de datos");
  e.printStackTrace();
}
```


JDBC Statement

La interfaz Java JDBC `Statement` se utiliza para ejecutar sentencias SQL en una base de datos relacional.

Obtenemos un JDBC `Statement` de una conexión JDBC. Una vez que tengamos una instancia de Java `Statement` se puede ejecutar una consulta de base de datos o una actualización de base de datos con ella.

JDBC Query

Las consultas a una base de datos usando JDBC se hacen enviando sentencias SQL a la base de datos con un objeto `Statement` que se crea a partir de una conexión abierta y nos devuelve un objeto `ResultSet` con la consulta.

```
Connection connection = DriverManager.getConnection(url);
Statement statement = connection.createStatement();
ResultSet resultset = statement.executeQuery("SELECT * FROM contactos");
```

JDBC Query

El objeto `ResultSet` devuelto está compuesto por filas (registros) con columnas de datos (campos).

```
while (resultset.next()) {
  String name = resultset.getString("nombre");
  int phone = resultset.getInt("telefono");
  String email = resultset.getString("email");
  System.out.println(name + "\t" + phone + "\t" + email);
}
```

JDBC Query

El método `next()` mueve el cursor a la siguiente fila y devuelve `true` si existe, `false` en caso contrario.

Debe llamarse al menos una vez, ya que antes de la primera llamada el cursor se coloca antes de la primera fila.

Los datos de columna para la fila actual se obtienen llamando a algunos de los métodos `getXXX()`, donde XXX es un tipo de datos primitivo.

Más información

JDBC Update

Para actualizar la base de datos hay que utilizar un objeto `Statement` como en la consulta, pero en lugar de llamar al método `executeQuery()` se llama al `executeUpdate()`.

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

String    sql       = "update people set name='John' where id=123";

int rowsAffected    = statement.executeUpdate(sql);
```

Más información

JDBC ResultSet

Un `ResultSet` es el resultado de una consulta de base de datos a través de un `Statement` o un `PreparedStatement`.

Cuando se crea un `ResultSet` hay tres atributos que se pueden establecer:

1. Type: cómo movernos por el ResultSet.
2. Concurrency: si podemos actualizar o solo leer los datos.
3. Holdability: si el ResultSet se cierra tras un commit().


Más información

JDBC PreparedStatement

Un `PreparedStatement` es un tipo especial de `Statement` con algunas características adicionales:

- Permite el uso parámetros en la instrucción SQL.
- Fácil de reutilizar con nuevos valores de parámetros.
- Puede aumentar el rendimiento de las sentencias ejecutadas.
- Permite actualizaciones por lotes más fáciles.


Más información

```
String sql = "update people set firstname=? , lastname=? where id=?";

PreparedStatement preparedStatement =
        connection.prepareStatement(sql);

preparedStatement.setString(1, "Gary");
preparedStatement.setString(2, "Larson");
preparedStatement.setLong  (3, 123);

int rowsAffected = preparedStatement.executeUpdate();

preparedStatement.setString(1, "Stan");
preparedStatement.setString(2, "Lee");
preparedStatement.setLong  (3, 456);

int rowsAffected = preparedStatement.executeUpdate();
```

Creación

Inserción de parámetros

Ejecución

Reutilización

JDBC Batch Updates

JDBC Batch Updates es un conjunto de actualizaciones agrupadas y enviadas a la base de datos en un lote. Es más rápido que enviarlas una a una, esperando que termine cada una:

- Menos tráfico de red en el envío de actualizaciones (solo un viaje de ida y vuelta).
- Se podrían ejecutar actualizaciones en paralelo.

Podemos usarlo con `Statement` y `PreparedStatement`.

JDBC Batch Updates

```
Statement statement = null;

try{
    statement = connection.createStatement();

    statement.addBatch("update people set firstname='John' where id=123");
    statement.addBatch("update people set firstname='Eric' where id=456");
    statement.addBatch("update people set firstname='May'  where id=789");

    int[] recordsAffected = statement.executeBatch();
} finally {
    if(statement != null) statement.close();
}
```

Añadimos al lote

Ejecutamos el lote

JDBC Transactions

Una *transacción* es un conjunto de acciones que se llevarán a cabo como una única acción atómica. O se realizan todas las acciones, o no se realiza ninguna.

La iniciamos con *connection*.`setAutoCommit(false)` y después ejecutamos las consultas y actualizaciones.

Si algo falla deshacemos todo con *connection*.`rollback()`;

Si hay todo bien confirmamos con *connection*.`commit()`;

Más información

```
try{
    connection.setAutoCommit(false);


    // perform operations on the JDBC Connection
    // which are to be part of the transaction

    connection.commit();
} catch (SQLException e) {
    connection.rollback();
}
```

JDBC CallableStatement

Un `CallableStatement` se usa para llamar a procedimientos almacenados en una base de datos.

Algunas operaciones pesadas de la base de datos pueden beneficiarse en cuanto al rendimiento si se ejecutan dentro del mismo espacio de memoria que el servidor de la base de datos, como un procedimiento almacenado.

Podemos pasarle parámetros y recoger los valores devueltos.

```
CallableStatement callableStatement =
    connection.prepareCall("{call calculateStatistics(?, ?)}");

callableStatement.setString(1, "param1");
callableStatement.setInt   (2, 123);

callableStatement.registerOutParameter(1, java.sql.Types.VARCHAR);
callableStatement.registerOutParameter(2, java.sql.Types.INTEGER);

ResultSet result = callableStatement.executeQuery();
while(result.next()) { ... }

String out1 = callableStatement.getString(1);
int    out2 = callableStatement.getInt   (2);
```


Más información

JDBC DatabaseMetaData

A través de la interfaz `DatabaseMetaData` se pueden obtener metadatos sobre la base de datos a la que nos hemos conectado.

Por ejemplo, podemos ver las tablas definidas en la base de datos, las columnas de cada tabla, si las características dadas son compatibles, etc.


Más información