

Atomicidad y JoinRowSet

Enlace al código fuente:

<https://github.com/arfloreshn/atomicidad>



JoinRowSet

Este método pasivo provisto por Oracle en su lenguaje Java SE provee la oportunidad de relacionar dos resultados.

JoinRowSet	Ejemplo
<pre>graph TD RowSet --> JdbcRowSet RowSet --> CachedRowSet CachedRowSet --> WebRowSet WebRowSet --> JoinRowSet WebRowSet --> FilteredRowSet</pre>	<pre>RowSetFactory factory = RowSetProvider.newFactory(); try (CachedRowSet coffees = factory.createCachedRowSet(); CachedRowSet suppliers = factory.createCachedRowSet(); JoinRowSet jrs = factory.createJoinRowSet()) { coffees.setCommand("SELECT * FROM COFFEES"); // Set connection parameters for the CachedRowSet coffees.execute(); suppliers.setCommand("SELECT * FROM SUPPLIERS"); // Set connection parameters for the CachedRowSet suppliers.execute(); jrs.addRowSet(coffees, "SUP_ID"); jrs.addRowSet(suppliers, "SUP_ID");</pre>

Puede resultar una opción de conveniente para quienes buscan trabajar con diferentes bases de datos distribuidas en diferentes servidores.

Cada responsable de proyecto deberá evaluar si es de conveniencia o no al costo de su proyecto implementar este método de programación y deberá evaluar en termino de complejidad y tiempo de implementación si este método es de suficiente valor agregado para lo que se necesita y busca como resultado final.

Aunque en la actualidad estos métodos son pocos conocidos y en su mayoría ensombrecidos por herramientas más robustas y completas como apache Kafka y su comunicación en tiempo real o los buses de servicios.

Siempre podemos abocarnos a estas utilidades cuando así se requiera, podemos abocarnos a estas soluciones que son más tradicionales o alternativas de menor coste para conectar dos o más servidores y volcar todos los resultados y realizar joins entre resultados.

La información de este artículo se tomó del libro Computing Distribuid Java 9, computación distribuida de Java, que en resumen son varios tópicos de diversas soluciones y herramientas para trabajar justamente temas relacionados con ambientes en distintos lugares físicos en una empresa.

Junto a los métodos antes expuesto tenemos los métodos de tipo pasivo de la api javax.sql.rowset, estos métodos pasivos o inactivos que puede ser de interés para quienes buscan soluciones a problemas de distribución de datos.

Métodos Pasivos de RowSet

Inteface	Descripción
CachedRowSet	Permite crear un cache de resultados de formas pasivas o desacoplada
JdbcRowSet	Permite crear consultas a base de datos
JoinRowSet	Permite crear relaciones entre resultados
RowSetFactory	Instancia una fábrica de RowSet
WebRowSet	Permite leer y escribir en un resultado XML
FilterRowSet	Permite el filtrado de resultado

Adicional a esas posibilidades, podríamos también añadir métodos de programación que mejore la atomicidad y reduzca la latencia de fallos en la red.

Ejemplo de uso de JoinRowSet:

En este ejemplo veremos cómo crear una relación entre dos resultados que estarán distintos servidores, aunque de antemano existen otros métodos de conexión entre bases de datos distribuidas como: dblink en Oracle y tablas Federated en Mysql.

Los siguientes ejemplos se enfocan para enseñar el principio de **ATOMICIDAD** y compartir el uso de la API **JoinRowSet** de Java y como relacionar resultados de distintos nodos de red o servidores de red.

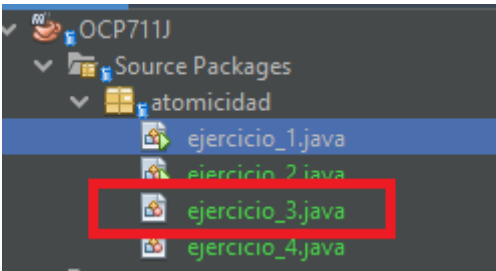
En los siguientes ejemplos vamos a consolidar todas las facturas de una agencia con la oficina central, para ello los campos item_sucursal y desc_agencia serán lo que estableceran la relación.

Estos ejercicios son para que el desarrollador tenga la seguridad de poder establecer relaciones entre dos resultados diferentes siempre y cuando los datos entre las columnas o campos a relacionar contengan datos que coincidan.

Explicaciones previas:

```
urlAgencia -> Es la cadena de conexión a la agencia.
urlCentral -> Es la cadena de conexión a la oficina central.
cnCentral -> Es la conexión a la oficina central o casa matriz.
cnAgencia -> Es la conexión a la agencia.
CacheRowSet cacheAgencia > sera el cache del resultado de las Facturas a consolidar.
CacheRowSet cacheOficinaCentral -> sera el cache del resultado de las agencias.
```

Contenido: Atomicidad

Ejercicio 1	Descripción
	<p>Contiene un programa que genera 1millon de facturas.</p> <p>Y realizaremos un único insert en la tabla de factura de la dbAgencia.</p>
Ejercicio 2	Descripción
	<p>Contiene un programa que consolidad las facturas en la oficina principal, mediante un solo insert, usando Statement() ,addbach() y executeBatch().</p> <p>Realizaremos un único insert en la dbCentral</p>
Ejercicio 3	Descripción
	<p>Contiene un programa que consolida las facturas en la oficina principal, mediante un solo insert, usando los métodos PreparedStatement(),addbach() y executeBatch().</p> <p>Realizaremos un único insert en la dbCentral.</p>
Ejercicio 4	Descripción
	<p>Contiene un programa que consolida las facturas en la oficina principal, mediante un solo insert, usando Statement() , Arreglo[] y executeBatch().</p> <p>Realizaremos un único insert en la dbCentral</p>

```
package atomicidad;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author AllanRamiro
 * @nacionalidad: Hondureña
 * @creado: 20/01/2021
 */
public class ejercicio_1 {

    public static void main(String[] args) throws ParseException, SQLException {

        int items = 1000000;
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd", Locale.ENGLISH);
        String urlAgencia = "jdbc:mysql://localhost:3306/dbAgencia?useServerPrepStmts=false&rewriteBatchedStatements=true";
        String user = "root"; //poner el usuario que tiene acceso a ambos host
        String pass = "root.8"; //poner la clave que tiene acceso a ambos host

        String dd = "";
        String mm = "";
        String yyyy = "2020";
        String fecha = "";
        String item_agencia = "";
        Date date = null;

        int dia = 1;
        int mes = 0;
        int nro_agencia = 1;
        double importe = 0;

        String arreglo[] = new String[items];
        for (int i = 0; i < items; i++) {

            dia = 1;
            mes = (int) (Math.random() * 11 + 0);
            nro_agencia = (int) (Math.random() * 140 + 1);
            item_agencia = "AGENCIA #" + String.valueOf(nro_agencia).trim();

            if (mes == 2) {
                dia = (int) (Math.random() * 28 + 1);
            } else {
                dia = (int) (Math.random() * 30 + 1);
            }

            dd = String.format("%02d", dia);
            mm = String.format("%02d", mes);

            date = formatter.parse(yyyy + "-" + mm.trim() + "-" + dd.trim());
            fecha = String.format("%1$tY-%1$tm-%1$te", date);

            importe = Math.round(Math.random() * (1000000 + 1000) * 100) / 100;
            arreglo[i] = "(" + fecha + "," + item_agencia + "," + importe + ")";
        }
    }
}
```

```
try {
    Class.forName("com.mysql.jdbc.Driver");

    Connection cnAgencia;
    cnAgencia = DriverManager.getConnection(urlAgencia, user, pass);
    cnAgencia.setAutoCommit(false);
    Statement st = cnAgencia.createStatement();

    String ins = "INSERT INTO facturas(fecha_factura, item_sucursal, imp_total_vta) VALUES";

    String data = String.format(" %s;", ins + String.join(" ", arreglo));

    System.out.println("Proceso Iniciado");
    st.executeUpdate(data);
    cnAgencia.setAutoCommit(true);
    System.out.println("Proceso finalizado");

} catch (ClassNotFoundException ex) {
    Logger.getLogger(ejercicio_1.class.getName()).log(Level.SEVERE, null, ex);
}
}
```

Ejercicio 2

```
package atomicidad;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.sql.rowset.CachedRowSet;
import javax.sql.rowset.JoinRowSet;
import javax.sql.rowset.RowSetFactory;
import javax.sql.rowset.RowSetProvider;

/**
 *
 * @author AllanRamiro
 */
public class ejercicio_2 {

    public static void main(String[] args) throws ClassNotFoundException {

        try {

            Class.forName("com.mysql.jdbc.Driver");

            String urlAgencia = "jdbc:mysql://localhost:3306/dbAgencia";
            String urlCentral = "jdbc:mysql://localhost:3306/dbCentral?useServerPrepStmts=false&rewriteBatchedStatements=
true";
            String user = "root"; //poner el usuario que tiene acceso a ambos host
            String pass = "root.8"; //poner la clave que tiene acceso a ambos host

            Connection cnAgencia;
            Connection cnCentral;

            cnAgencia = DriverManager.getConnection(urlAgencia, user, pass);
            cnCentral = DriverManager.getConnection(urlCentral, user, pass);

            Statement agencia = cnAgencia.createStatement();
            Statement OficinaCentral = cnCentral.createStatement();

            String consulta = "SELECT fecha_factura, item_sucursal,imp_total_vta FROM facturas "
                    + "WHERE fecha_factura between '2020-06-01' and '2020-06-30'";

            ResultSet rsFacturas = agencia.executeQuery(consulta);
```

```
ResultSet rsAgencias = OficinaCentral.executeQuery("SELECT agencia_id,desc_agencia FROM agencias");
```

```
RowSetFactory factory = RowSetProvider.newFactory();
```

```
CachedRowSet cacheAgencia = factory.createCachedRowSet();
```

```
CachedRowSet cacheOficinaCentral = factory.createCachedRowSet();
```

```
cacheAgencia.populate(rsFacturas); //Instanciamos los cache del reporte o la data a importar
```

```
cacheOficinaCentral.populate(rsAgencias); //Instanciamos el cache de las sucursales
```

```
/*----- ALGORITMO FORK-JOIN -----*/
```

Algoritmo Fork-Join

En este patrón de diseño se generan dos ejecuciones concurrentes,

que empieza inmediatamente después de que el fork es llamado en código,

después se usa join para combinar estas dos ejecuciones concurrentes en una.

Cada join puede unirse entonces a su fork correspondiente y lo hace antes de las otras terminen.

EL API JoinRowSet de Java proporciona la aplicación del algoritmo Fork-Join

CON ESTE ALGORITMO CREAREMOS UNA RELACION ENTRE LOS CACHE

PREVIAMENTE CARGADOS, NO ENTRE TABLAS, LAS TABLAS PUEDEN ESTAR EN "N" SERVIDOR

```
-----*/
```

```
JoinRowSet jrs = factory.createJoinRowSet(); // Declaramos una variable de para relacionar los caches con data
que cargamos previamente
```

```
jrs.addRowSet(cacheAgencia, "item_sucursal"); // Establecemos los campos que vamos a relacion en este caso
es "item_sucursal"
```

```
jrs.addRowSet(cacheOficinaCentral, "desc_agencia"); // Establecemos los campos que vamos a relacion en es
te caso es "desc_agencia"
```

```
/*-----
```

METODO 2, Statement - API JODBC JAVA

```
-----*/
```

```
Statement cmd_uno = cnCentral.createStatement();
```

```
cnCentral.setAutoCommit(false);
```

```
jrs.beforeFirst(); //Mueve el cursor al primer registro
```

```
int counter = 0;
```

```
while (jrs.next()) {
```

```
String sql = "INSERT INTO RESUMEN_FACTURAS(fecha,agencia_id,importe)"
```

```
+ " VALUES (" + jrs.getString("fecha_factura").substring(0, 10) + "," + jrs.getInt("agencia_id") + "," + jrs.get
```

```
Double("imp_total_vta") + ")";
```

```
cmd_uno.addBatch(sql);
```

```
counter++;
```

```
//al tener 1000 registros, mandamos todas a ejecutar el insert
```

```
if (counter == 1000) {
```

```
cmd_uno.executeBatch();
```

```
counter = 0;
```

```
}
```

```
    }

    //revisamos si todavía hay sentencias pendientes de ejecutar
    if (counter > 0) {
        cmd_uno.executeBatch();
    }

    cnCentral.setAutoCommit(true);
    System.out.println("Fin del medicion Bloque 1:" + new Date());

} catch (SQLException ex) {
    Logger.getLogger(ejercicio_2.class.getName()).log(Level.SEVERE, null, ex);
}

}

}
```

Ejercicio 3

```
package atomicidad;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.sql.rowset.CachedRowSet;
import javax.sql.rowset.JoinRowSet;
import javax.sql.rowset.RowSetFactory;
import javax.sql.rowset.RowSetProvider;

/**
 *
 * @author AllanRamiro
 */
public class ejercicio_3 {

    public static void main(String[] args) throws ClassNotFoundException {

        try {

            Class.forName("com.mysql.jdbc.Driver");
```



```

String urlAgencia = "jdbc:mysql://localhost:3306/dbAgencia";
String urlCentral = "jdbc:mysql://localhost:3306/dbCentral?useServerPrepStmts=false&rewriteBatchedStatements=
true";

String user = "root"; //poner el usuario que tiene acceso a ambos host
String pass = "root.8"; //poner la clave que tiene acceso a ambos host

Connection cnAgencia;
Connection cnCentral;

cnAgencia = DriverManager.getConnection(urlAgencia, user, pass);
cnCentral = DriverManager.getConnection(urlCentral, user, pass);

Statement agencia = cnAgencia.createStatement();
Statement OficinaCentral = cnCentral.createStatement();

String consulta = "SELECT fecha_factura, item_sucursal,imp_total_vta FROM facturas "
+ "WHERE fecha_factura between '2020-04-01' and '2020-04-30'";

ResultSet rsFacturas = agencia.executeQuery(consulta);
ResultSet rsAgencias = OficinaCentral.executeQuery("SELECT agencia_id,desc_agencia FROM agencias");

RowSetFactory factory = RowSetProvider.newFactory();
CachedRowSet cacheAgencia = factory.createCachedRowSet();
CachedRowSet cacheOficinaCentral = factory.createCachedRowSet();

cacheAgencia.populate(rsFacturas); //Instanciamos los cache del reporte o la data a importar
cacheOficinaCentral.populate(rsAgencias); //Instanciamos el cache de las sucursales

/*----- ALGORITMO FORK-JOIN -----*/
Algoritmo Fork-Join
En este patrón de diseño se generan dos ejecuciones concurrentes,
que empieza inmediatamente después de que el fork es llamado en código,
después se usa join para combinar estas dos ejecuciones concurrentes en una.
Cada join puede unirse entonces a su fork correspondiente y lo hace antes de las otras terminen.

EL API JoinRowSet de Java proporciona la aplicación del algoritmo Fork-Join
CON ESTE ALGORITMO CREAREMOS UNA RELACION ENTRE LOS CACHE
PREVIAMENTE CARGADOS, NO ENTRE TABLAS, LAS TABLAS PUEDEN ESTAR EN "N" SERVIDOR

-----*/
JoinRowSet jrs = factory.createJoinRowSet(); // Declaramos una variable de para relacionar los caches con data
que cargamos previamente
jrs.addRowSet(cacheAgencia, " item_sucursal"); // Establecemos los campos que vamos a relacion en este cas
o es "item_sucursal"
jrs.addRowSet(cacheOficinaCentral, "desc_agencia"); // Establecemos los campos que vamos a relacion en es
te caso es "desc_agencia"

/*-----*/
METODO 3, PreparedStatement - API JODBC JAVA
-----*/
//Desactivamos en autocommit lo vamos a confirmas hasta el final bloque
cnCentral.setAutoCommit(false);
PreparedStatement cmd_dos;
cmd_dos = cnCentral.prepareStatement("INSERT INTO RESUMEN_FACTURAS(fecha,agencia_id,importe) VALU
ES (?, ?, ?)");

int counter = 0;
jrs.beforeFirst();
System.out.println("Inicio del Bloque 2:" + new Date());
while (jrs.next()) {

    cmd_dos.setString(1, jrs.getString("fecha_factura").substring(0, 10));
    cmd_dos.setInt(2, jrs.getInt("agencia_id"));
    cmd_dos.setDouble(3, jrs.getDouble("imp_total_vta"));
    cmd_dos.addBatch();
    counter++;
}

```

```
//al tener 1000 registros, mandamos todas a ejecutar el insert
if (counter == 1000) {
    cmd_dos.executeBatch();
    counter = 0;
}

}

//revisamos si todavía hay sentencias pendientes de ejecutar
if (counter > 0) {
    cmd_dos.executeBatch();
}

cnCentral.setAutoCommit(true);
System.out.println("Fin de la respuesta Bloque 2:" + new Date());

} catch (SQLException ex) {
    Logger.getLogger(ejercicio_3.class.getName()).log(Level.SEVERE, null, ex);
}

}

}
```

Ejercicio_4

```
package atomicidad;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.sql.rowset.CachedRowSet;
import javax.sql.rowset.JoinRowSet;
import javax.sql.rowset.RowSetFactory;
import javax.sql.rowset.RowSetProvider;

/**
 *
 * @author AllanRamiro
 */
public class ejercicio_4 {

    public static void main(String[] args) throws ClassNotFoundException {

        try {

            Class.forName("com.mysql.jdbc.Driver");

            //363577
            String urlAgencia = "jdbc:mysql://localhost:3306/dbAgencia";
            String urlCentral = "jdbc:mysql://localhost:3306/dbCentral?useServerPrepStmts=false&rewriteBatchedStatements=
true";
            String user = "root"; //poner el usuario que tiene acceso a ambos host
            String pass = "root.8"; //poner la clave que tiene acceso a ambos host

            Connection cnAgencia;
            Connection cnCentral;

            cnAgencia = DriverManager.getConnection(urlAgencia, user, pass);
            cnCentral = DriverManager.getConnection(urlCentral, user, pass);

            Statement agencia = cnAgencia.createStatement();
            Statement OficinaCentral = cnCentral.createStatement();

            String consulta = "SELECT fecha_factura, item_sucursal,imp_total_vta FROM facturas "
                    + "WHERE fecha_factura between '2020-01-01' and '2020-02-28'";

            ResultSet rsFacturas = agencia.executeQuery(consulta);
            ResultSet rsAgencias = OficinaCentral.executeQuery("SELECT agencia_id,desc_agencia FROM agencias");

            RowSetFactory factory = RowSetProvider.newFactory();
            CachedRowSet cacheAgencia = factory.createCachedRowSet();
            CachedRowSet cacheOficinaCentral = factory.createCachedRowSet();

            cacheAgencia.populate(rsFacturas); //Instanciamos los cache del reporte o la data a importar
            cacheOficinaCentral.populate(rsAgencias); //Instanciamos el cache de las sucursales
```

```
/*----- ALGORITMO FORK-JOIN -----*/
```

Algoritmo Fork-Join
En este patrón de diseño se generan dos ejecuciones concurrentes, que empieza inmediatamente después de que el fork es llamado en código, después se usa join para combinar estas dos ejecuciones concurrentes en una. Cada join puede unirse entonces a su fork correspondiente y lo hace antes de las otras terminen.

EL API JoinRowSet de Java proporciona la aplicación del algoritmo Fork-Join
CON ESTE ALGORITMO CREAREMOS UNA RELACION ENTRE LOS CACHE
PREVIAMENTE CARGADOS, NO ENTRE TABLAS, LAS TABLAS PUEDEN ESTAR EN "N" SERVIDOR

```
-----*/
```

```
JoinRowSet jrs = factory.createJoinRowSet(); // Declaramos una variable de para relacionar los caches con data
que cargamos previamente
jrs.addRowSet(cacheAgencia, " item_sucursal"); // Establecemos los campos que vamos a relacion en este caso
es "item_sucursal"
jrs.addRowSet(cacheOficinaCentral, "desc_agencia"); // Establecemos los campos que vamos a relacion en este
caso es "desc_agencia"
```

```
/*-----
```

```
METODO 4 - Statement - API JODBC JAVA
```

```
-----*/
```

```
cnCentral.setAutoCommit(false);
Statement st = cnCentral.createStatement();
String insert = "INSERT INTO RESUMEN_FACTURAS(fecha,agencia_id,importe) VALUES";

int counter = 0;
String filas[] = new String[999];
String enviarInsert[] = new String[counter];
System.out.println("Inicio del Bloque 3:" + new Date());
jrs.beforeFirst();
while (jrs.next()) {
    counter++;
    enviarInsert = new String[counter];

    filas[counter - 1] = "(" + jrs.getString("fecha_factura").substring(0, 10) + "," + jrs.getString("agencia_id") + "," + jrs.getDouble("imp_total_vta") + ")";
    System.arraycopy(filas, 0, enviarInsert, 0, counter); // Redimensionar Arreglo conservando los datos

    //al tener 1000 registros, mandamos todas a ejecutar el insert masivo para evitar mucha recarga en la base de datos
    //y reiniciamos el contador
    if (counter == 999) {
        counter = 0;
        //Esta instruccion concatena la variable insert, mas(+) la conversion del arreglo transformado y agregar ; al final de formateo de la concatenación
        String ins = String.format(" %s;", insert + String.join(", ", enviarInsert));
        st.executeUpdate(ins);
        enviarInsert = new String[counter];
        filas = new String[999];
    }
}

//revisamos si todavía hay sentencias pendientes de ejecutar
if (counter > 0) {
    String ins = String.format(" %s;", insert + String.join(", ", enviarInsert));
    st.executeUpdate(ins);
}

cnCentral.setAutoCommit(true);
System.out.println("Fin del Bloque 3:" + new Date());

} catch (SQLException ex) {
    Logger.getLogger(ejercicio_4.class.getName()).log(Level.SEVERE, null, ex);
}
```

```
}  
}
```

Conclusión

Todos métodos reducen el tiempo de respuesta de IO en las bases de datos y las peticiones al servidor, todos los métodos se ejecutarán un único insert, los cuatro(4) métodos antes expuestos cumplen con el criterio de atomicidad y reducen la latencia y tráfico de red.

La **atomicidad** es la propiedad que asegura que una operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias. **Se dice que una operación es atómica cuando es imposible para otra parte de un sistema encontrar pasos intermedios.** Si esta operación consiste en una serie de pasos, todos ellos ocurren o ninguno. Por ejemplo, en el caso de una transacción bancaria o se ejecuta tanto el depósito y la deducción o ninguna acción es realizada. Es una característica de los sistemas transaccionales.

Puede le interés explorar los siguientes enlaces:

7.2. Remote Database Applications

https://docstore.mik.ua/orelly/java-ent/dist/ch07_02.htm

7.3. Multi-Database Applications

https://docstore.mik.ua/orelly/java-ent/dist/ch07_03.htm

Sumado a esta lista de opciones podemos también agregar los siguientes métodos de programación: