Introducción

En este laboratorio, vamos a intentar comprender cómo funciona el sistema de control de concurrencia de *MySQL*. Para esto, implementaremos un cliente que realizará distintas operaciones de manera concurrente en la base de datos y comprobaremos si el funcionamiento es el esperado o no.

Objetivos

Los objetivos de este laboratorio son:

Aprender a utilizar la herramienta JMeter para realizar pruebas en un servidor MySQL.

Comprender los mecanismos de control de concurrencia para gestionar transacciones en MySQL.

Comprobar la dificultad de gestionar el acceso concurrente a la información.

Contenido

Este laboratorio está dividido en 2 partes:

1. Apache JMeter: presentación y guía de uso de la herramienta.
2. Control de concurrencia: mecanismos de gestión de la concurrencia en MySQL. Esta parte incluye un ejercicio a realizar por parejas.

Requisitos previos

Para realizar este laboratorio utilizaremos el servidor de bases de datos que obtuvimos como resultado del Laboratorio 1. El servidor tiene que estar correctamente configurado para ser accesible remotamente.

También es necesario tener un entorno de ejecución Java en nuestro equipo local, al igual que se utilizó en el Laboratorio 2.

1 Apache JMeter

En esta primera sección se presenta Apache JMeter, la herramienta que se utilizará a lo largo del laboratorio para trabajar con el control de concurrencia en MySQL. Utilizaremos JMeter desde nuestro equipo local, para trabajar contra nuestro servidor.

JMeter es una herramienta de la fundación Apache orientada a realizar pruebas de carga y rendimiento en diferentes tipos de servicios, como p.e. bases de datos compatibles con JDBC, servicios Web o servidores LDAP, entre otros. Es una herramienta muy utilizada en el sector industrial por su madurez y modularidad, ya que es compatible con *plug-ins* de 3ºs que extienden su funcionalidad.

Su última versión es la 5.4.3. Se puede obtener más información sobre JMeter en su web oficial, incluyendo documentación y ejemplos de uso: https://jmeter.apache.org/

1.1 Instalación

Para poder ejecutar JMeter en **nuestro equipo local**, es necesario tener una instalación funcional de Java (bien sea JRE o JDK). Esto se puede comprobar con el siguiente comando desde un terminal:

```
:~$ java -version
```

```
unai@MBP ~ % java -version
java version "1.8.0_221"
Java(TM) SE Runtime Environment (build 1.8.0_221-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.221-b11, mixed mode)
```

Si el comando anterior devuelve un nº de versión y algo más de información sobre la instalación de Java, es correcto. No es necesario tener el mismo número de versión que aparece en la imagen para hacer el laboratorio. Si en lugar de un número de versión aparece un error, revisar la instalación de Java.

Una vez comprobado que Java funciona correctamente, el siguiente paso es descargar la última versión de JMeter desde la siguiente dirección y descomprimir el archivo en una carpeta de nuestro equipo:

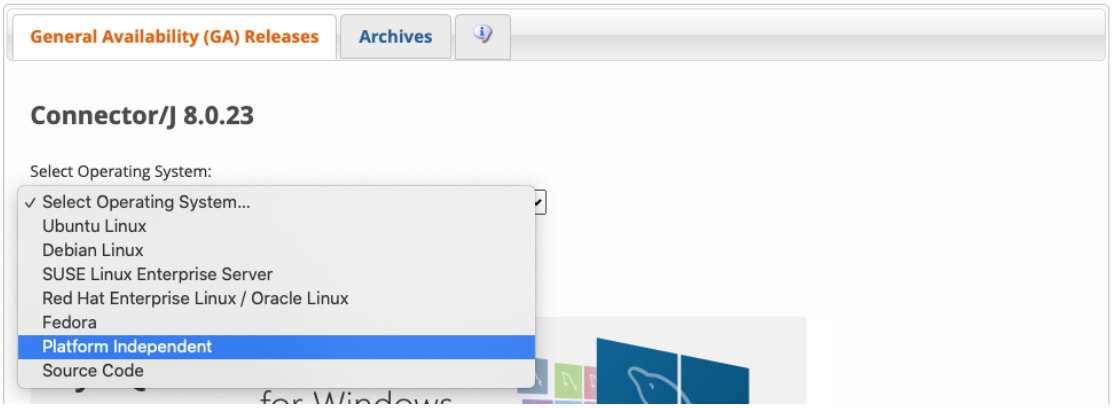
https://dlcdn.apache.org//jmeter/binaries/apache-jmeter-5.4.3.tgz

Tras descomprimir el archivo, se genera una carpeta llamada *apache-jmeter-5.4.3* con el siguiente contenido:

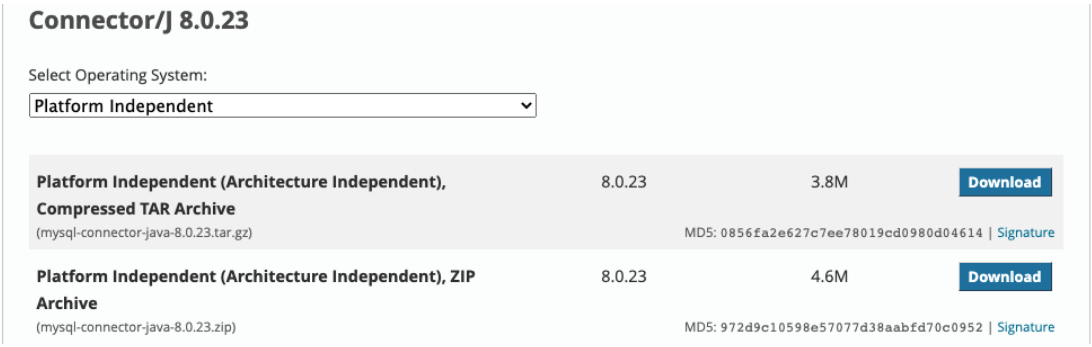
```
unai@MBP ~ % ls -l apache-jmeter-5.4.1
total 64
-rw-r--r--    1 unai  staff  15292  2 ene  1970 LICENSE
-rw-r--r--    1 unai  staff    167  2 ene  1970 NOTICE
-rw-r--r--    1 unai  staff   9821  2 ene  1970 README.md
drwxr-xr-x   43 unai  staff   1376  2 ene  1970 bin
drwxr-xr-x    6 unai  staff    192  2 ene  1970 docs
drwxr-xr-x   22 unai  staff    704  2 ene  1970 extras
drwxr-xr-x  104 unai  staff   3328  2 ene  1970 lib
drwxr-xr-x  104 unai  staff   3328  2 ene  1970 licenses
drwxr-xr-x   19 unai  staff    608  2 ene  1970 printable_docs
```

Con este paso se han obtenido los ficheros principales para ejecutar JMeter, pero es necesario descargar un fichero más que permita a la herramienta conectarse a bases de datos MySQL: el conector JDBC.

Entrar en la web del conector MySQL (https://dev.mysql.com/downloads/connector/j/) y descargar la versión "independiente de plataforma", tal y como se indica en la siguiente imagen:



Se mostrará la siguiente web, ofreciendo la descarga en 2 tipos de formatos comprimidos, TAR o ZIP:



Ambos ficheros contienen el mismo conector, elegir la descarga en el formato más apropiado para vuestro sistema. En caso de duda, elegir el formato ZIP.

¡ATENCIÓN!

Descargar el archivo con el conector y descomprimirlo. Se genera una carpeta llamada *mysql-connector-java-8.0.23* con el siguiente contenido:

```
unai@MBP ~ % ls -l mysql-connector-java-8.0.23
total 5648
-rw-r--r--@ 1 unai  staff   267994  1 dic 17:29 CHANGES
-rw-r--r--@ 1 unai  staff      186  1 dic 17:29 INFO_BIN
-rw-r--r--@ 1 unai  staff      136  1 dic 17:29 INFO_SRC
-rw-r--r--@ 1 unai  staff   100771  1 dic 17:29 LICENSE
-rw-r--r--@ 1 unai  staff     1245  1 dic 17:29 README
-rw-r--r--@ 1 unai  staff    88878  1 dic 17:29 build.xml
-rw-r--r--@ 1 unai  staff  2415211  1 dic 17:29 mysql-connector-java-8.0.23.jar
drwxr-xr-x@ 8 unai  staff      256  1 dic 17:29 src
```

Copiar el archivo “jar” incluido en la carpeta del conector dentro del directorio “lib” de JMeter:

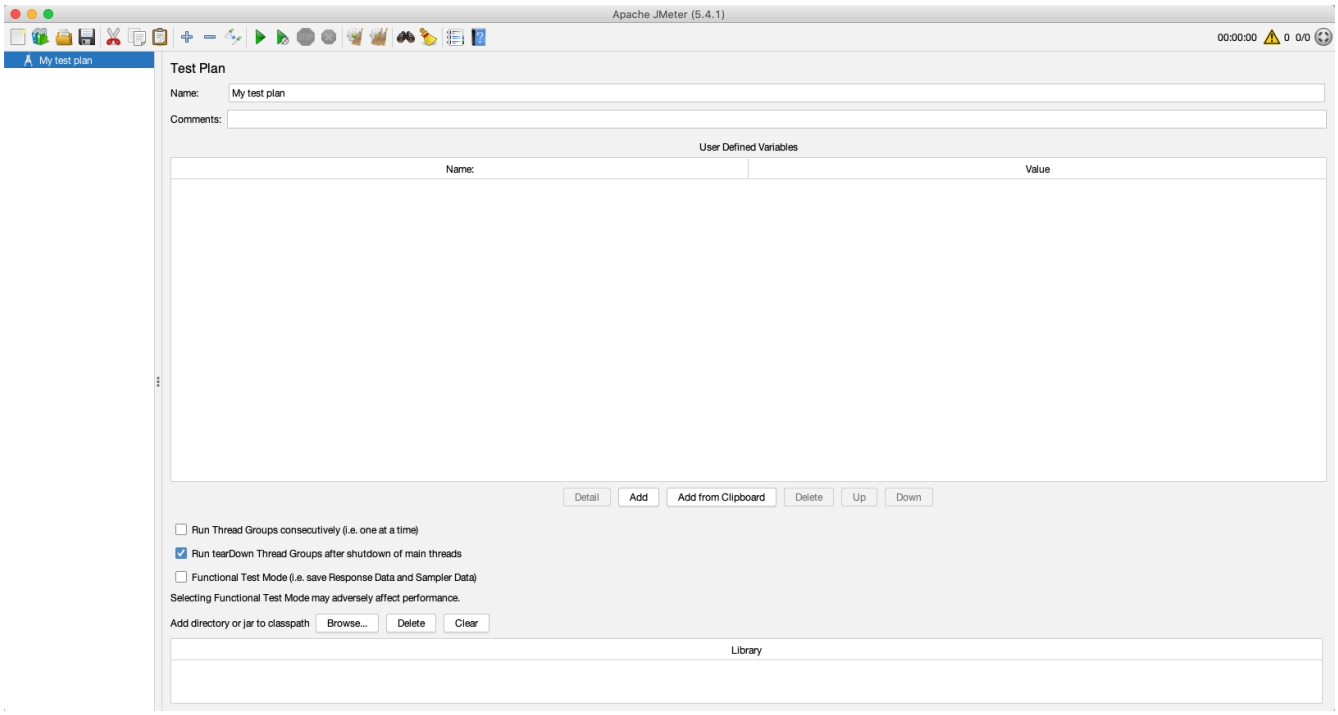
```
unai@MBP ~ % cp mysql-connector-java-8.0.23/mysql-connector-java-8.0.23.jar apache-jmeter-5.4.1/lib
unai@MBP ~ % ls -lh apache-jmeter-5.4.1/lib/mysql-connector-java-8.0.23.jar
-rw-r--r--@ 1 unai  staff   2,3M  9 abr 11:42 apache-jmeter-5.4.1/lib/mysql-connector-java-8.0.23.jar
```

Hecho esto, JMeter está listo para ser usado con un servidor MySQL. Se puede iniciar desde un

terminal indicando la ruta al "jar" de JMeter (la ruta puede variar en vuestro equipo):

```
:~$ java -jar apache-jmeter-5.4.1/bin/ApacheJMeter.jar
```

Tras unos segundos, se lanzará la interfaz visual de JMeter. En la imagen se muestra con el tema visual "Darklaf - IntelliJ". Podéis cambiar el tema visual en el menú "Options", sección "Look & feel":



1.2 Uso básico

En esta sección se detallan los conceptos básicos para utilizar JMeter y se propone un ejercicio para ponerlos en práctica.

Para poder realizar éste parte del laboratorio, es necesario tener en marcha nuestro servidor MySQL y algunos datos de muestra. Para esto último, se provee este fichero: https://github.com/ulopeznovoa/ABD-DBK-Lab5-Files/blob/main/loadPeopleData.sql

Este fichero es un script SQL que crea lo siguiente en MySQL:

- Una nueva BBDD llamada "JMeterTest".
- Una tabla "People" en "JMeterTest" con las siguientes columnas: "id" (entero, clave primaria auto incremental), "name" (varchar de 50 caracteres) y "surname" (varchar de 100 caracteres).
- Los datos ficticios (nombre y apellido) de 5 personas en la tabla "People".
- Un usuario "JMeterUser" con contraseña "mysql2021" y permisos "select" y "update" en la

tabla “People”.

Antes continuar con JMeter, abrid una sesión SSH con vuestro servidor (o instancia GCP) y cargad los datos de prueba en MySQL utilizando el usuario “root” y el archivo “loadPeopleData.sql” indicado en el repositorio:

¡ATENCIÓN! antes de ejecutar el comentado; para que el mencionado script funcione de manera correcta, vuestra política de password en mysql debe estar configurada como LOW. Así, en primer lugar comprobad que eso es así, y si no lo es, cambiad para que el script se ejecute correctamente.

En Mysql (como root)

```
1-SHOW VARIABLES LIKE 'validate_password_policy';  (Verificación)

2- SET GLOBAL validate_password_policy = LOW; (Cambio en caso necesario)
```

Después, cargar los datos de prueba en MySQL , utilizando el usuario “root”:

```
nagorebarrenaehu@nago-ubuntu:~$ wget https://github.com/ulopeznovoa/ABD-DBK-Lab5-Files/blob/ma
in/loadPeopleData.sql
--2022-03-17 09:56:42--  https://github.com/ulopeznovoa/ABD-DBK-Lab5-Files/blob/main/loadPeopl
eData.sql
Resolving github.com (github.com)... 140.82.121.3
Connecting to github.com (github.com)|140.82.121.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'loadPeopleData.sql'

loadPeopleData.sql          [ <=>                       ] 132.19K  --.-KB/s    in 0.02s

2022-03-17 09:56:42 (7.30 MB/s) - 'loadPeopleData.sql' saved [135359]

nagorebarrenaehu@nago-ubuntu:~$
```

Una vez cargado el script, si no hay señales de error, es recomendable abrir la consola de MySQL y verificar que la tabla “People” se ha creado correctamente y que el usuario “JMeterUser” pueda leerla.

Una vez que los datos se han cargado, debéis leer el siguiente tutorial de la documentación de JMeter para entender cómo se configura y ejecuta un plan de pruebas contra MySQL. Un plan de pruebas (o Test Plan) es una configuración que incluye una serie de consultas SQL y la forma en la que se van a lanzar (cuántas veces, si la van a lanzar múltiples hilos, etc). Podéis utilizar los datos que acabáis de cargar para probar los conceptos que se explican en el tutorial:

https://jmeter.apache.org/usermanual/build-db-test-plan.html

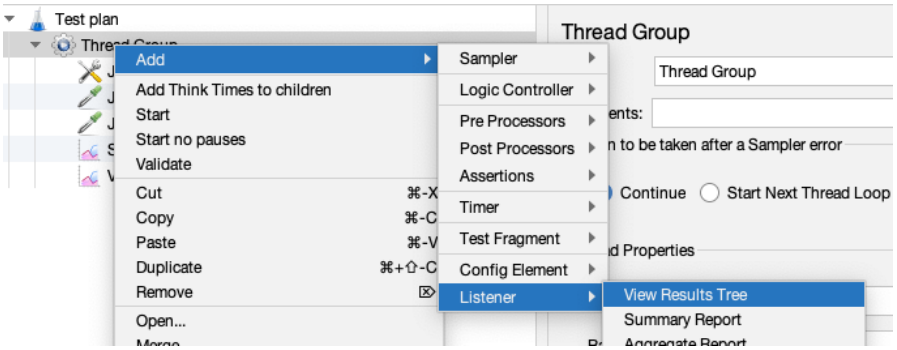
Mientras utilicéis JMeter, se puede configurar el idioma en el menú "Options" (sección "Choose language"). También se puede activar el log de errores y avisos en el menú "Options" (opción "Log viewer") para mostrar ver la actividad e inspeccionar posibles problemas.

Una vez leído el tutorial y entendido el funcionamiento de JMeter, se propone el siguiente ejercicio:

Crear un Test Plan con las siguientes características. Se deberá configurar en JMeter y debe utilizar los datos de la BBDD "JMeterTest":

- *1 único hilo (o usuario).*
- *1 JDBC Request llamado "People data" que obtenga todos los datos de la tabla "People".*
- *1 JDBC Request llamado "People count" que cuente el número de filas de la tabla "People".*

Para poder visualizar el resultado de las consultas, hay que añadir el tipo de Listener "View Results Tree" al Test Plan, el cual no se describe en el tutorial indicado más arriba. Este Listener contiene el resultado de cada JDBC Request en su sección "Response Data".



El resultado del ejercicio debería ser el siguiente: Al hacer click en el Listener "Summary report", se muestran las estadísticas sobre cada una de las conexiones "JDBC Request" (los valores de las columnas "Average" a "Avg. Bytes" en la imagen pueden no coincidir con los vuestros):

Summary Report

Name: Summary Report

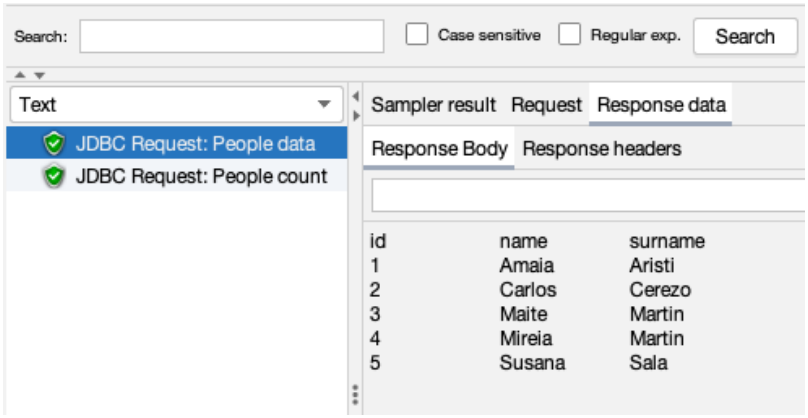
Comments:

Write results to file / Read from file

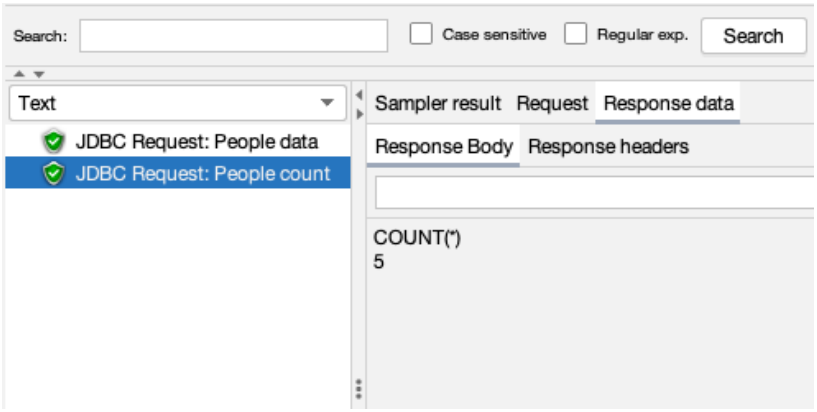
Filename Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
JDBC Request: People data	1	2176	2176	2176	0,00	0,00%	27,6/min	0,04	0,00	92,0
JDBC Request: People count	1	113	113	113	0,00	0,00%	8,8/sec	0,10	0,00	11,0
TOTAL	2	1144	113	2176	1031,50	0,00%	52,4/min	0,04	0,00	51,5

Por otra parte, al hacer click en el Listener "View Results Tree", se pueden ver los resultados de cada una de las JDBC Request. En la siguiente figura se muestra el resultado para "People data":



En la siguiente figura se muestra para "People count":



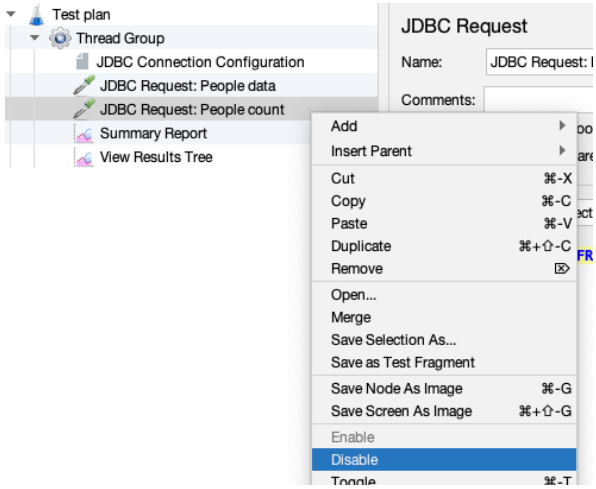
1.3 Configuración específica

En esta sección se van a detallar algunas características específicas sobre JMeter que serán útiles para desarrollar la segunda parte de este laboratorio. Para cada una de ellas se plantea junto con un breve

ejercicio que ayudará a ponerla en práctica, utilizando el ejercicio de la sección anterior como base.

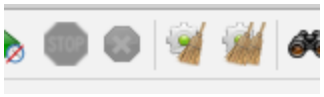
1.3.1 Activar y desactivar módulos

Es posible habilitar y deshabilitar módulos concretos de un Test Plan para evitar que se ejecuten sin tener que eliminarlos. Se consigue con las opciones Enable y Disable del menú contextual que aparece al hacer click derecho sobre cualquiera de los elementos.



Ejercicio: deshabilitar el JDBC Request “People count” y verificar que en los Listeners sólo aparecen los resultados del “People data”.

Para limpiar los resultados y registros de ejecuciones anteriores y que sólo aparezcan los últimos, se pueden utilizar los botones con iconos de escoba que se encuentran en la botonera superior. El botón de la izquierda limpia los resultados del módulo elegido, mientras que el botón de la derecha elimina de todos.



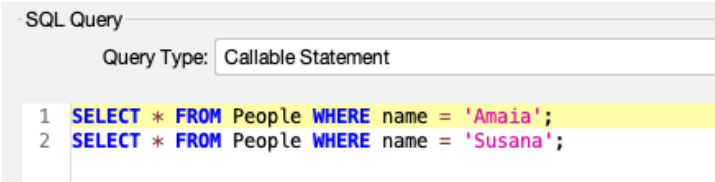
1.3.2 Múltiples sentencias en un JDBC Request

Por defecto, JMeter sólo permite que se ejecute una única sentencia SQL en un JDBC Request. Sin embargo, es posible escribir conjuntos de varias sentencias en un único JDBC Request de la siguiente

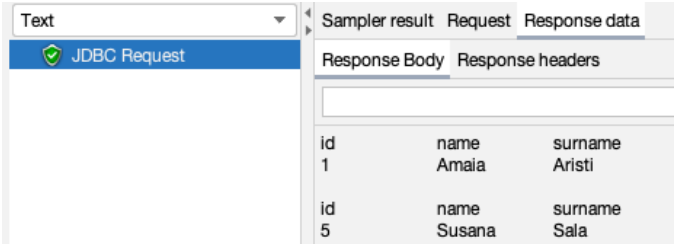
forma.

Añadir el sufijo "?allowMultiQueries=true" a la URL del servidor en el módulo "Configuración de la conexión JDBC":

Después, en cada JDBC Request que queramos escribir múltiples sentencias SQL, utilizar el tipo de consulta "Callable Statement":



Al hacerlo así, los resultados en un listener "View Result Tree" aparecerán agrupados. En la imagen, el resultado de ejecutar la consulta anterior:

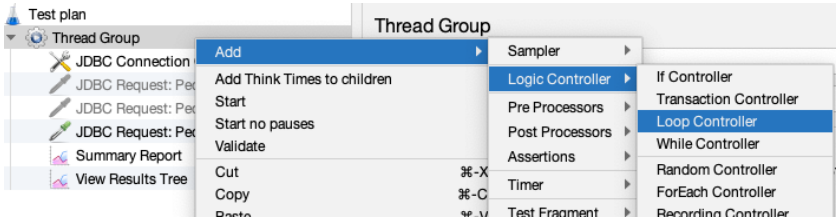


Ejercicio: crear un nuevo JDBC Request llamado "People data & count" que ejecute las consultas SQL utilizadas en los JDBC Requests "People data" y "People count".

1.3.3 Controladores

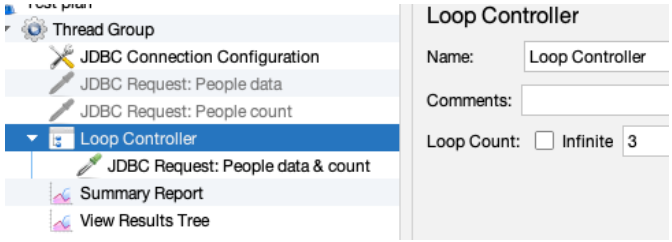
JMeter proporciona múltiples módulos llamados “controladores” que sirven para organizar módulos que ejecutan acciones (como los JDBC Request). Utilizando controladores podemos indicar que un módulo concreto se ejecute condicionalmente, en bucle, o de cualquier otra forma.

Todos los controladores disponibles se encuentran en el menú contextual que aparece al hacer click derecho sobre un grupo de hilos:



Como ejemplo concreto, para conseguir que un módulo se repita varias veces podemos utilizar el “Loop Controller”, que hace que los hilos ejecuten uno o varios módulos en bucle. Para ello, añadir un “Loop Controller” desde el menú indicado arriba y configurar en la nueva ventana el número de veces que queremos que se repita en el campo “Loop count”:

Después, mover el controlador al punto que creamos conveniente dentro del Test Plan y arrastrar a su interior los módulos que queramos que se ejecuten en bucle. En la imagen inferior, el Loop Controller ejecutaría 3 veces el JDBC Request “People data & count”:



Ejercicio: Crear un Test Plan con un Loop Controller que ejecute los JDBC Requests de “People data” y “People count” 20 veces. Verificar que se han ejecutado correctamente en la sección de listeners.

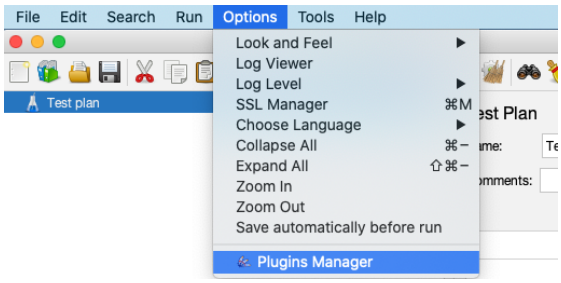
1.3.4 Ejecutar JDBC Requests en paralelo

A pesar de que JMeter incluye varios tipos de controladores en su instalación por defecto, ninguno de ellos permite que se ejecuten varios módulos o JDBC Requests diferentes en paralelo. Esto será necesario para el ejercicio de la 2ª parte del Laboratorio y, por ello, en esta sección veremos cómo incluir un controlador de 3ºs en JMeter que provea esta funcionalidad.

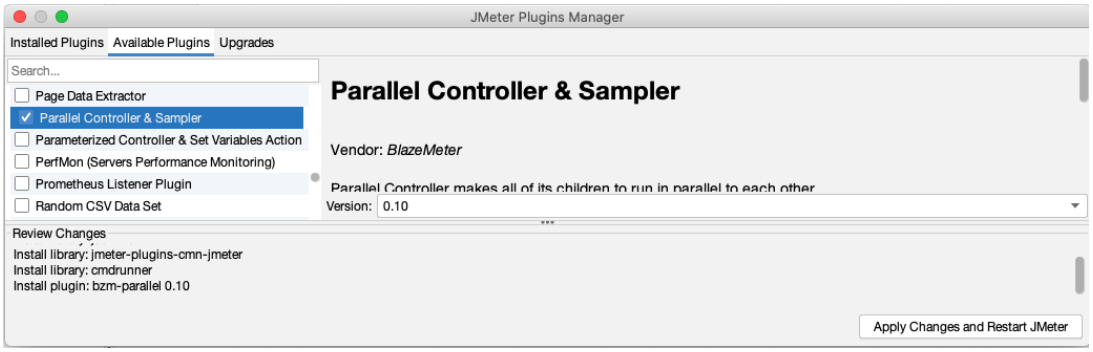
El primer paso es instalar el gestor de *plugins* de JMeter. Si tenéis JMeter en marcha, cerrad, descargad el fichero *jmeter-plugins-manager-1.6.jar* de esta dirección (https://jmeter-plugins.org/get/) y moverlo a la carpeta lib/ext dentro del directorio de JMeter:

```
unai@MBP ~ % ls apache-jmeter-5.4.1/lib/ext/jmeter-plugins-manager-1.6.jar
apache-jmeter-5.4.1/lib/ext/jmeter-plugins-manager-1.6.jar
```

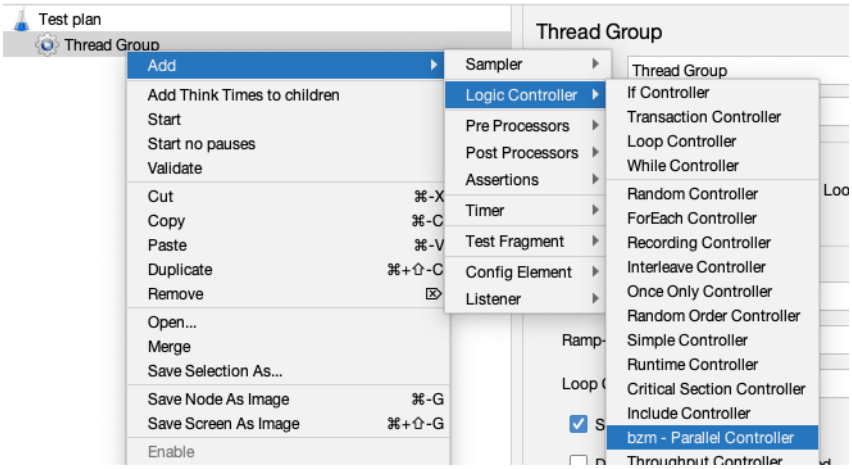
Iniciar JMeter de nuevo y abrir el gestor de *plugins*. Se encuentra al final del menú “Options”:



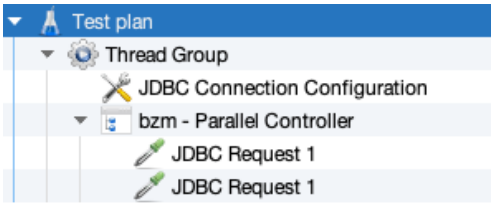
Desde la sección “Available plugins”, buscar y seleccionar el *plugin* “Parallel Controler & Sampler”. Hacer click en “Apply changes and Restart JMeter” y esperar unos segundos a que JMeter se reinicie:



Tras el reinicio de JMeter, el módulo "Parallel Controller" estará disponible dentro del menú de controladores.



El "Parallel Controller" lanza un hilo por cada módulo que contenga anidado para que la ejecución de cada uno de ellos comience simultáneamente. En la imagen de ejemplo, el controlador tiene 2 JDBC Requests:



Si lanzásemos el plan de prueba de la imagen con 2 hilos en el "Thread Group" y los 2 JDBC Requests dentro del controlador, se iniciarían 4 hilos en ejecución simultáneamente. Cada hilo del Thread Group ejecutaría un "Parallel Controller", y cada uno de ellos a su vez lanzaría otros 2 hilos, uno para cada JDBC Request.

Ejercicio: Reemplazar el "Loop Controller" utilizado en el ejercicio anterior por un "Parallel Controller" que ejecute los JDBC Requests "People data" y "People count" en paralelo. Utilizar 1 único hilo en el Thread Group.

2. Control de concurrencia en MySQL

Tras haber aprendido a utilizar JMeter, en este apartado se presenta en detalle el sistema de gestión de concurrencia y transacciones en MySQL, y posteriormente se plantea un ejercicio a realizar en parejas.

IMPORTANTE: Antes de continuar, se recomienda encarecidamente desactivar todos los *logs* de MySQL que se hayan activado para realizar laboratorios o ejercicios anteriores (incluyendo el log binario). Las instrucciones para desactivar los diferentes *logs* se pueden encontrar en los Laboratorios 3 y 4 de la asignatura.

2.1 Niveles de aislamiento

MySQL ofrece varios niveles de aislamiento para tratar la consistencia en transacciones:

1. *Repeatable read*: Es el nivel por defecto. Si en una misma transacción se realizan varias consultas "Select", se asegura la consistencia entre ellas.
2. *Read committed*: Es un nivel menos estricto que *Repeatable read*. Es apropiado para situaciones donde obtener resultados estrictamente repetibles es menos importante que obtener mayor rendimiento (por el menor número de bloqueos).
3. *Read uncommitted*: Es un nivel menos estricto que *Read uncommitted*. Si se realizan varias consultas "Select" en una misma transacción, estas se ejecutan sin realizar bloqueos y podrían resultar en lecturas de filas no actualizadas.
4. *Serializable*: Es un nivel más estricto que *Repeatble read.* En una transacción con varias consultas "Select", se trata cada "Select" como una transacción individual. Se utiliza en situaciones que requieren alta fiabilidad o para depurar problemas de concurrencia.

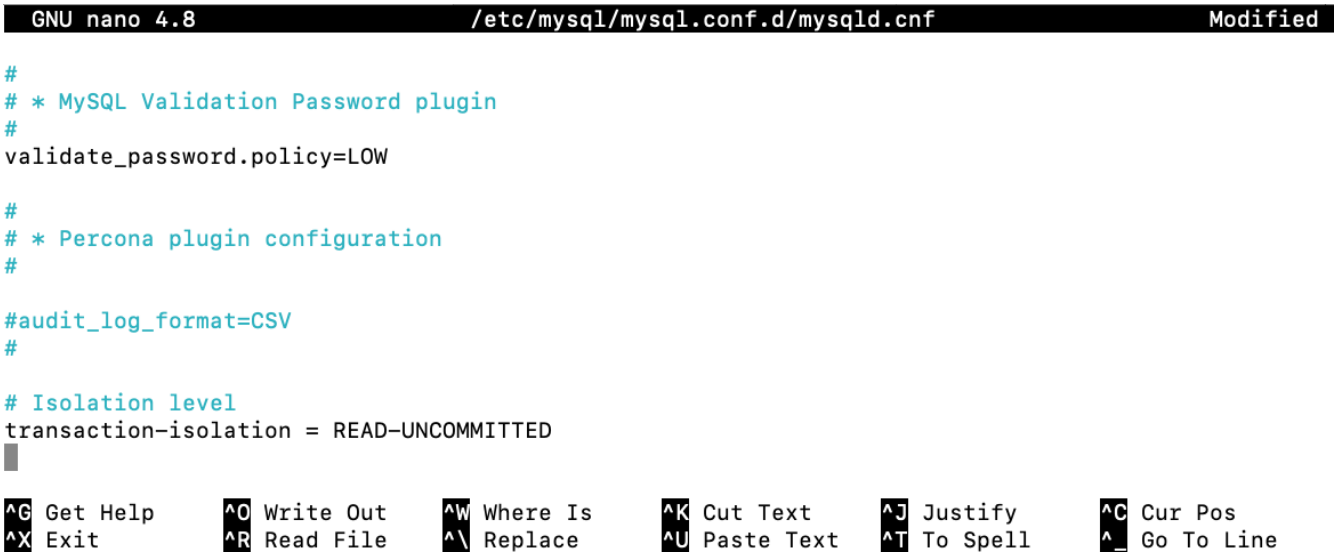
Se puede obtener más información y ejemplos sobre estos niveles de aislamiento en la siguiente sección de la documentación: https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html

MySQL ofrece varias formas de cambiar el nivel de aislamiento en un servidor, pero en este laboratorio sólo veremos una: mediante una variable en el fichero de configuración.

Abrir el fichero de configuración de MySQL y añadir la siguiente línea:

```
# Isolation level
transaction-isolation = READ-UNCOMMITTED
```

Esta línea configura el nivel de aislamiento como *Read uncommited*. Para establecer otros niveles de aislamiento, utilizar uno de los siguientes valores: READ-COMMITTED, REPEATABLE-READ o SERIALIZABLE.



Después de modificar el fichero de configuración, reiniciar el servicio MySQL para que el cambio surta efecto. Una vez reiniciado, el nivel de aislamiento se puede consultar desde la consola:

```
mysql> select @@TRANSACTION_ISOLATION;
```

```
mysql> select @@TRANSACTION_ISOLATION;
+-------------------------+
| @@TRANSACTION_ISOLATION |
+-------------------------+
| READ-UNCOMMITTED        |
+-------------------------+
1 row in set (0.00 sec)
```

Como vamos a trabajar con 2 servidores distintos, podéis configurar cada servidor MySQL con un nivel de aislamiento diferente.

2.2 Transacciones

Por defecto, cada sentencia que ejecutamos en MySQL (bien a través de la consola o con un script pasado por parámetro) se ejecuta en una transacción individual. Esto sucede porque, por defecto, el modo *autocommit* está activado, es decir, MySQL incluye una sentencia *commit* después de cada sentencia y cada cambio se registra de inmediato (si no ha habido error). En caso de error, MySQL decide si aplicar *commit* o *rollback* en función del error. Se puede obtener más información sobre este comportamiento en: https://dev.mysql.com/doc/refman/8.0/en/innodb-autocommit-commit-rollback.html

En este laboratorio trabajaremos con transacciones sin *autocommit* para explorar de forma manual la gestión de cambios y de concurrencia. Hay varias formas de trabajar sin *autocommit*, la que utilizaremos en este laboratorio es la creación de transacciones de forma manual dentro de procedimientos almacenados. Esto nos permitirá tener mayor control sobre la gestión de *commit* y *rollback*.

En este laboratorio, todo procedimiento almacenado tendrá una estructura basada en la siguiente:

```
CREATE PROCEDURE NombreProcedimiento(parámetros)
BEGIN
  DECLARE VariablesLocales
  DECLARE FLAG BOOL DEFAULT 0;
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET FLAG = 1;
  DECLARE CONTINUE HANDLER FOR SQLWARNING SET FLAG = 1;

  REPEAT
    SET FLAG = 0;
    START TRANSACTION;

    Sentencia 1;
    Sentencia 2;
    …

    IF FLAG THEN
      ROLLBACK;
    ELSE
      COMMIT;
```

```
    END IF;
  UNTIL NOT FLAG END REPEAT;
END;
```

En los siguientes párrafos se explican las sentencias SQL utilizadas para construir esta estructura.En MySQL, el comienzo de una transacción individual se indica con la sentencia START TRANSACTION. Al ejecutar esta sentencia, si el modo *autocommit* está activado, se desactiva hasta que se ejecuta una sentencia COMMIT o ROLLBACK que indique si los cambios deben hacerse persistentes o descartarse.

En este laboratorio vamos a implementar procedimientos que, en caso de que fallen, se reinicien automáticamente. Para ello, utilizamos las sentencias DECLARE de SQL, las cuales tienen la siguiente estructura:

DECLARE <acción >HANDLER FOR <situación> <sentencia(s)>

Las sentencias DECLARE dentro de un procedimiento almacenado indican qué <acción> hacer cuando sucede <situación>, y además permiten ejecutar <sentencia(s)> antes de realizar <acción>. En nuestra estructura, estas sentencias se utilizan para que, en caso de excepción (error) o aviso (warning), se ponga a 1 (“verdadero”) la variable booleana “Flag”. Después, en caso de error/aviso, la transacción continua hasta el bloque “if” final donde, si ha habido una excepción/aviso SQL, se hará un ROLLBACK. En caso contrario, se hará un COMMIT. La variable “Flag” también se utiliza para controlar el bucle REPEAT que repetirá la transacción se ejecute sin excepciones ni errores.

Para la definición de procedimientos almacenados, consideraremos que se podrán hacer, al menos, las siguientes sentencias:

- Lecturas (READ): Serán sentencias “Select” para las cuales habrá que decidir el modo de bloqueo de lectura. Se debe consultar la siguiente sección de la documentación: https://dev.mysql.com/doc/refman/8.0/en/innodb-locking-reads.html
- Escrituras (WRITE): Serán sentencias “Update” normales.

Para el contexto de este laboratorio, se recomienda realizar las operaciones aritméticas necesarias utilizando los recursos que proporciona SQL, utilizando variables locales dentro de procedimientos. En el siguiente fragmento se muestra un ejemplo de cómo declarar una variable local de tipo entero, utilizarla para guardar un valor de una tabla e incrementar su valor en 1.

```
DECLARE var int;
…
SELECT columna INTO var FROM table WHERE …;
SELECT var + 1 INTO var;
```

3. Ejercicio

Utilizando todo lo aprendido hasta ahora, se plantea el siguiente ejercicio a realizar en grupos de 2 estudiantes.

Cada estudiante tendrá que implementar 3 de los siguientes procedimientos (Bloque A o B). Después creará un Test Plan con JMeter para ejecutar 100 veces cada uno de los procedimientos del bloque, de manera paralela, primero contra su propio servidor, después contra el servidor de su pareja:

Bloque A

Procedure A	Procedure B	Procedure C
READ(X)	READ(Y)	READ(Z)
X = X+1	Y = Y+1	Z = Z+1
WRITE(X)	WRITE(Y)	WRITE(Z)
READ(T)	READ(T)	READ(T)
READ(A)	READ(B)	READ(C)
READ(Y)	READ(Z)	READ(X)
T = T+Y	T = T+Z	T = T+X
A = A+Y	B = B+Z	C = C+X
WRITE(T)	WRITE(T)	WRITE(T)
WRITE(A)	WRITE(B)	WRITE(C)

Bloque B

Procedure D	Procedure E	Procedure F
READ(T)	READ(T)	READ(T)
READ(D)	READ(E)	READ(F)
READ(Z)	READ(X)	READ(Y)
T = T+Z	T = T+X	T = T+Y
D = D+Z	E = E+X	F = F+Y
WRITE(T)	WRITE(T)	WRITE(T)
WRITE(D)	WRITE(E)	WRITE(F)
READ(X)	READ(Y)	READ(Z)
X = X-1	Y = Y-1	Z = Z-1
WRITE(X)	WRITE(Y)	WRITE(Z)

Para poder realizar este ejercicio, es necesario que cada miembro de la pareja tenga en marcha su servidor MySQL y que además tenga unos datos concretos. Para esto último, se provee el siguiente script SQL:
https://github.com/ulopeznovoa/ABD-DBK-Lab5-Files/blob/main/loadConcurrencyData.sql

Este script crea lo siguiente en MySQL:

- Una nueva BBDD llamada "concurrency".
- Una tabla "variables" en "concurrency" con las siguientes columnas: "name" (varchar de 1 caracter) y "value" (entero).
- 12 filas en la tabla "variables", cada una con una letra en "name" (X, Y, Z, A, B, C, D, E, F, T) y el número 0 en "value".
- Un usuario "concurrencyUser" con contraseña "dba-2021" que dispone de permisos "select" y "update" en la tabla "variables" y permisos para crear, modificar y ejecutar procedimientos almacenados a nivel de sistema.
- Un procedimiento almacenado "CheckResults" que valida los resultados del ejercicio.

Antes de comenzar con el ejercicio se debe cargar el script en MySQL utilizando el usuario "root" (tal y como se hizo en la 1ª parte de este laboratorio). Una vez cargado, si no hay señales de error, es recomendable abrir la consola de MySQL y verificar que la tabla "variables" se ha creado correctamente y que el usuario "concurrencyUser" puede leerla.

El resultado final esperado (es decir, correcto) tras ejecutar los 6 procedimientos en mismo servidor sobre el conjunto de datos indicado es el siguiente (independientemente del número de iteraciones):

> Variables "*X*", "*Y*", "*Z*": 0.
>
> Variables "*A*", "*B*", "*C*", "*D*", "*E*", "*F*": cualquier valor.
>
> Variable "*T*": igual a la suma de "*A*", "*B*", "*C*", "*D*", "*E*" y "*F*".

Se debe verificar que la ejecución de los procedimientos da el mismo resultado con los diferentes niveles de aislamiento de MySQL.

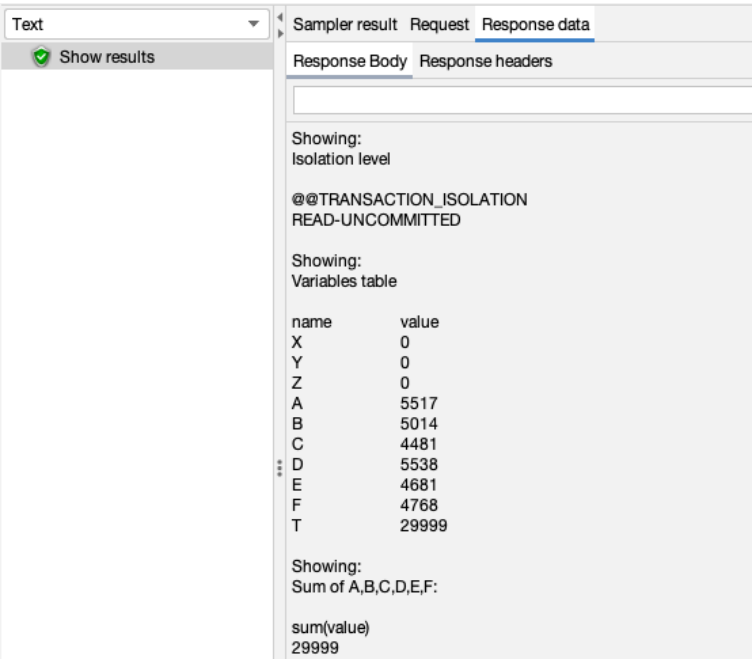
Se debe utilizar JMeter para implementar el ejercicio. Se recomienda crear diferentes Test Plans, cada uno con un objetivo diferente:

- Crear los procedimientos almacenados.
- Ejecutar los procedimientos almacenados en bucle.
- Visualizar el contenido de la tabla "variables" y verificar los valores de "X", "Y", "Z" y "T".

Se debe entregar en *eGela* uno o varios fichero .jmx que contengan los Test Plan realizados en JMeter. También se pueden entregar ficheros adicionales si fuese necesario.

En la siguiente imagen se muestra el resultado de la función CheckResults en un Listener de tipo "View Result Tree", que muestra el resultado correcto del ejercicio.

- Nivel de aislamiento en marcha (*Read Uncommited* en este caso).
- Contenido de la tabla "variables", con los valores indicados más arriba.
- Suma de los valores de "A", "B", "C", "D", "E" y "F.



En la siguiente imagen, en cambio, se muestra un resultado incorrecto para el ejercicio: "Z" no es 0 y la suma de los valores de "A" a "F" no son iguales al valor de "T":