

## 一、实验目的

1. 验证所学理论，巩固所学知识并加深理解；
2. 培养学生设计数据库压力测试分析的能力；
3. 熟悉 JMeter 中添加和管理插件的基本操作；
4. 熟悉 JMeter 中设置 JDBC 进行数据库测试的过程和操作。

## 二、实验项目内容

- 1、用实验环境中的 MySQL 数据库，创建 sakila 数据库，并运行 sakila.sql 脚本以创建相关数据库表
  - 2、在 JMeter 中建立数据库 JDBC 测试，指向本地的 sakila 数据库
  - 3、用 sql 语句访问 film 用户表，然后建立对应的数据库 select count(\*)脚本，运行 5 次，观察 select 脚本的访问时间
  - 4、用 sql 语句往 actor 用户表，然后建立对应的数据库 select count(\*)脚本，运行 5 次，观察 select 脚本的访问时间
  - 5、用 sql 语句往 rental 用户表，然后建立对应的数据库 select count(\*)脚本，运行 5 次，观察 select 脚本的访问时间
  - 6、用 sql 语句实现对演员和主演电影的组合访问，要求输出电影的基本信息和主要演员的姓名，建立对应的数据库 select 脚本，运行 5 次，观察 select 脚本的访问时间
  - 7、用 sql 语句实现对 CD 租赁情况的组合访问，要求输出店名、店出租的影片名称、租碟子的顾客基本信息、工作人员信息、出租时间和费用以及该影片的主演姓名。建立对应的数据库 select 脚本，运行 5 次，观察 select 脚本的访问时间
  - 8、在该 sakila 数据库中增加一个 member 表（memberid-自增类型，name-可以固定不变）。然后用 insert 命令，往该表中插入 1000 条记录。观察 insert 脚本执行的时长
  - 9、对 sakila 数据库中的 member 表做 select \* 查询，记录查询时长
  - 10、对 sakila 数据库中的 zc\_member 表做 select \* 查询，记录查询时长
- 要求：

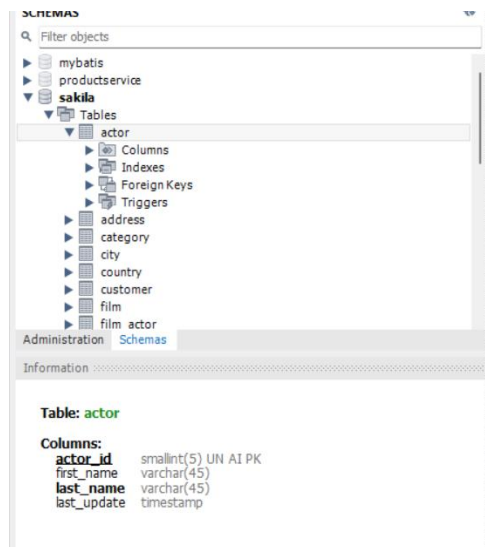
1. 提交上述相关操作的 JMeter 截图

2. 比较第 9 和第 10 步的时长，并简单说明时长区别分析

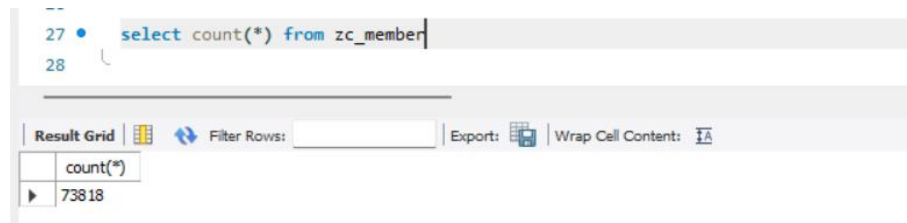
### 三、实验过程或算法（源程序）

#### 1. 用实验环境中的 MySQL 数据库，创建 sakila 数据库，并运行 sakila.sql 脚本以创建相关数据库表

本实验环境采用的是 MySQL5.5 版本,mysql-connector-java-5.1.42 驱动, jmeter5.6.3. 将实验提供的数据导入 workbench 得到相关的数据库表。



对数据库进行简单查询测试，可见配置正确。



#### 2. 在 JMeter 中建立数据库 JDBC 测试，指向本地的 sakila 数据库

将 jdbc 驱动导入/lib 库，并对 MySQL 数据库的 URL、username 和 password 进行配置，



#### 3. 用 sql 语句访问 film 用户表,然后建立对应的数据库 select count(\*)脚本,

## 运行 5 次，观察 select 脚本的访问时间

- 配置运行参数 5 次(下同，省略):



● 继续 ○ 启动下一进程循环 ○ 停止线程 ○ 停止测试 ○ 立即停止测试

线程属性

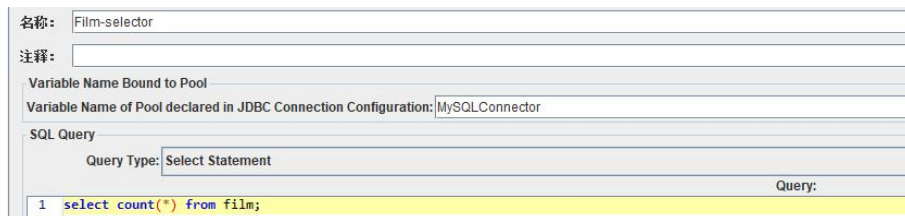
线程数: 1

Ramp-Up时间(秒): 1

循环次数 ☐ 永远 5

☒ Same user on each iteration

- 编写 SQL 语句:



名称: Film-selector

注释:

Variable Name Bound to Pool

Variable Name of Pool declared in JDBC Connection Configuration: MySQLConnector

SQL Query

Query Type: Select Statement

Query:

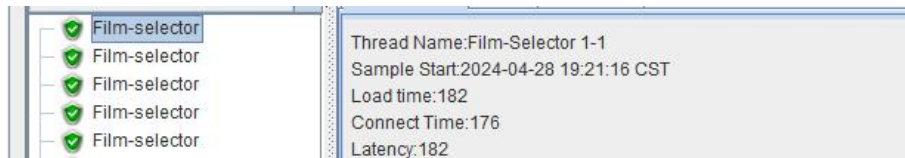
```
1 select count(*) from film;
```

- 测试结果:

第一次到第五次 Load time 分别为: 182, 12, 0, 1, 5

第一次到第五次 Latency 分别为: 182, 12, 0, 0, 5

第一次到第五次 Connect Time 分别为: 176, 0, 0, 0, 0



Film-selector

Film-selector

Film-selector

Film-selector

Film-selector

Thread Name: Film-Selector 1-1

Sample Start: 2024-04-28 19:21:16 CST

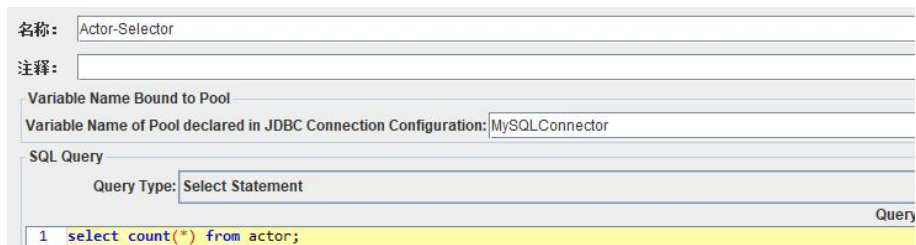
Load time: 182

Connect Time: 176

Latency: 182

## 4. 用 sql 语句往 actor 用户表，然后建立对应的数据库 select count(\*)脚本，运行 5 次，观察 select 脚本的访问时间

- 编写 SQL 语句:



名称: Actor-Selector

注释:

Variable Name Bound to Pool

Variable Name of Pool declared in JDBC Connection Configuration: MySQLConnector

SQL Query

Query Type: Select Statement

Query:

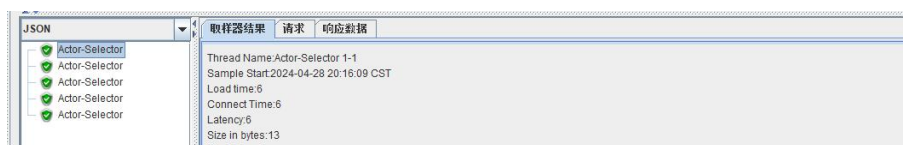
```
1 select count(*) from actor;
```

- 测试结果:

第一次到第五次 Load time 分别为: 6, 0, 0, 1, 0

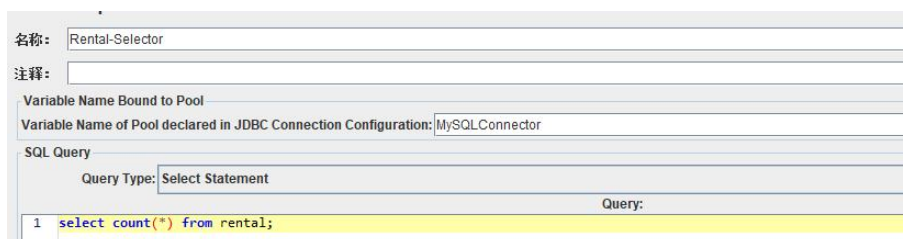
第一次到第五次 Latency 分别为: 6, 0, 0, 1, 0

第一次到第五次 Connect Time 分别为: 6, 0, 0, 0, 0



5. 用 sql 语句往 rental 用户表,然后建立对应的数据库 select count(\*)脚本,运行 5 次,观察 select 脚本的访问时间

- 编写 SQL 语句:

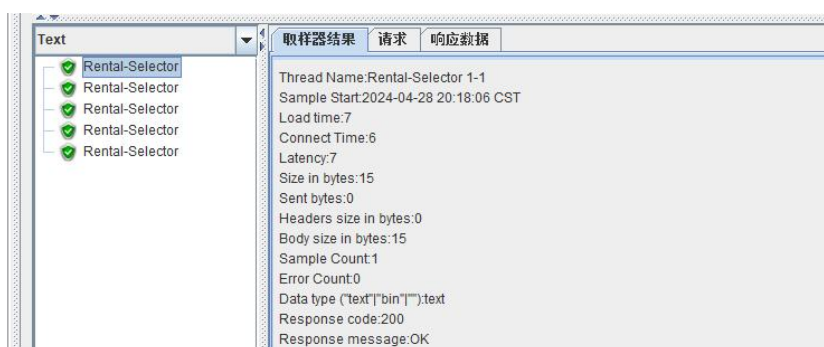


- 测试结果:

第一次到第五次 Load time 分别为: 7, 0, 0, 1, 0

第一次到第五次 Latency 分别为: 7, 0, 0, 0, 0

第一次到第五次 Connect Time 分别为: 6, 0, 0, 0, 0



6. 用 sql 语句实现对演员和主演电影的组合访问, 要求输出电影的基本信息和主要演员的姓名, 建立对应的数据库 select 脚本, 运行 5 次, 观察 select 脚本的访问时间

- 编写 SQL 语句:

名称: Actor-And-film-Selector

注释:

Variable Name Bound to Pool

Variable Name of Pool declared in JDBC Connection Configuration: MySQLConnector

SQL Query

Query Type: Select Statement

Query:

```

1 select * from actor a, film b, film_actor c
2 where a.actor_id = c.actor_id and b.film_id = c.film_id;

```

• 测试结果:

第一次到第五次 Load time 分别为: 170, 97, 83, 85, 72

第一次到第五次 Latency 分别为: 51, 48, 52, 56, 43

第一次到第五次 Connect Time 分别为: 8, 0, 0, 0, 0

Text

取样器结果 请求 响应数据

Actor-And-film-Selector

Actor-And-film-Selector

Actor-And-film-Selector

Actor-And-film-Selector

Actor-And-film-Selector

Thread Name: Actor-And-film-Selector 1-1

Sample Start: 2024-04-28 20:20:33 CST

Load time: 170

Connect Time: 8

Latency: 51

Size in bytes: 1487243

Sent bytes: 0

Headers size in bytes: 0

Body size in bytes: 1487243

Sample Count: 1

Error Count: 0

Data type ("text"|"bin"|): text

Response code: 200

Response message: OK

7. 用 sql 语句实现对 CD 租赁情况的组合访问, 要求输出店名、店出租的影片名称、租碟子的顾客基本信息、工作人员信息、出租时间和费用以及该影片的主演姓名。建立对应的数据库 select 脚本, 运行 5 次, 观察 select 脚本的访问时间

• 编写 SQL 语句:

名称: CD-Selector

注释:

Variable Name Bound to Pool:

Variable Name of Pool declared in JDBC Connection Configuration: MySQLConnector

SQL Query

Query Type: Select Statement

Query:

```

1 select store.store_id, payment.amount, rental.rental_date,
2     film.title, customer.first_name, customer.first_name, customer.last_name ,staff.first_name, staff.last_name,
3     actor.first_name, actor.last_name from()
4     ((((((payment
5     left join rental on payment.rental_id = rental.rental_id)
6     left join inventory on rental.inventory_id = inventory.inventory_id)
7     left join store on inventory.store_id = store.store_id)
8     left join film on inventory.film_id = film.film_id)
9     left join customer on payment.customer_id = customer.customer_id)
10    left join staff on payment.staff_id = staff.staff_id)
11    left join film_actor on inventory.film_id = film_actor.film_id)
12    left join actor on film_actor.actor_id = actor.actor_id)
13 )

```

- 测试结果:

第一次到第五次 Load time 分别为: 1351, 1364, 1362, 1508, 1363

第一次到第五次 Latency 分别为: 1104, 1142, 1162, 1268, 1175

第一次到第五次 Connect Time 分别为: 8, 0, 0, 0, 0

CD-Selector	Thread Name:CD-Selector 1-1
CD-Selector	Sample Start:2024-04-28 20:22:32 CST
CD-Selector	Load time:1351
CD-Selector	Connect Time:8
CD-Selector	Latency:1104
CD-Selector	Size in bytes:8045763
	Sent bytes:0
	Headers size in bytes:0
	Body size in bytes:8045763
	Sample Count:1
	Error Count:0
	Data type ("text" "bin"):"text"
	Response code:200
	Response message:OK

8. 在该 sakila 数据库中增加一个 member 表 (memberid-自增类型, name-可以固定不变)。然后用 insert 命令, 往该表中插入 1000 条记录。观察 insert 脚本执行的时长

- 创建表:

```

21 -- CREATE TABLE member (
22 --     memberid INT AUTO_INCREMENT PRIMARY KEY,
23 --     name VARCHAR(255) NOT NULL
24 -- );
25 • select * from member;
26
27
--

```

Result Grid

memberid	name
1	NULL

- 编写 SQL 语句, 重复 1000 次

**JDBC Request**

名称: Insert-Member

注释:

Variable Name Bound to Pool

Variable Name of Pool declared in JDBC Connection Configuration: MySQLConnector

SQL Query

Query Type: Update Statement

Query:

```
1 INSERT INTO member (name) VALUES ('leomooore');
2
```

- 测试结果:

一条记录的 Load time 和 Latency 均在 20 左右, 除第一次的 Connect Time 为 8, 其余均为 0.

Text	取样器结果	请求	响应数据
<ul style="list-style-type: none"> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> <li>Insert-Member</li> </ul>	<p>Thread Name: Insert-Member 1-1</p> <p>Sample Start: 2024-04-28 20:27:30 CST</p> <p>Load time: 18</p> <p>Connect Time: 0</p> <p>Latency: 18</p> <p>Size in bytes: 9</p> <p>Sent bytes: 0</p> <p>Headers size in bytes: 0</p> <p>Body size in bytes: 9</p> <p>Sample Count: 1</p> <p>Error Count: 0</p> <p>Data type ("text" "bin" ""): text</p> <p>Response code: 200</p> <p>Response message: OK</p> <p>SampleResult fields:</p> <p>ContentType: text/plain</p> <p>DataEncoding: UTF-8</p>		

## 9. 对 sakila 数据库中的 member 表做 select \* 查询, 记录查询时长

- 编写 SQL 语句:

**JDBC Request**

名称: Member-Selector

注释:

Variable Name Bound to Pool

Variable Name of Pool declared in JDBC Connection Configuration: MySQLConnector

SQL Query

Query Type: Select Statement

Query:

```
1 select * from member;
```

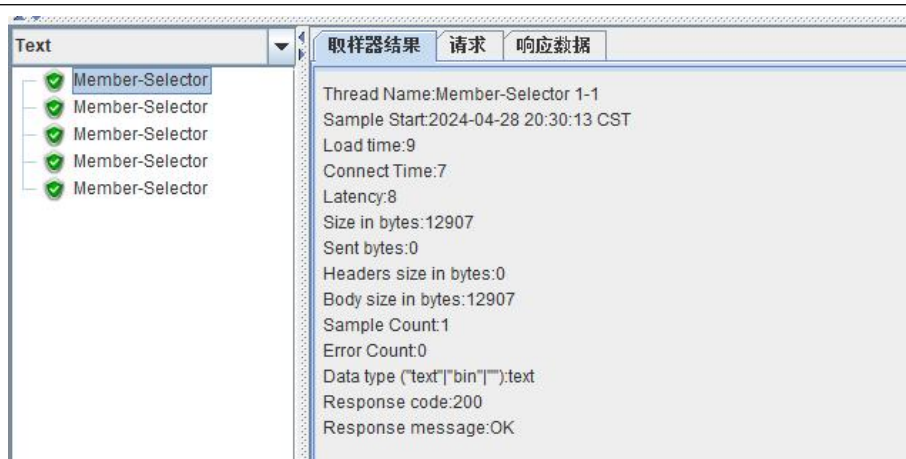
- 测试结果:

第一次到第五次 Load time 分别为: 9, 2, 1, 1, 1

第一次到第五次 Latency 分别为: 8, 1, 1, 1, 1

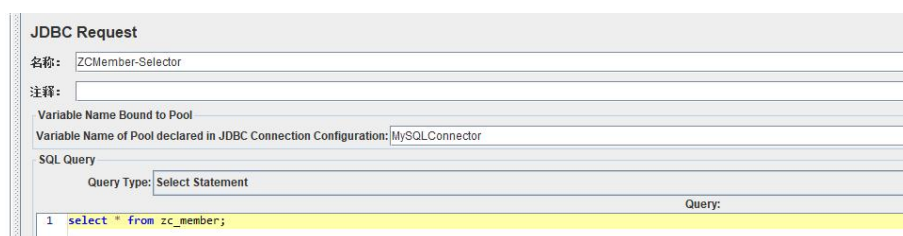
第一次到第五次 Connect Time 分别为: 7, 0, 0, 0, 0





## 10. 对 sakila 数据库中的 zc\_member 表做 select \* 查询，记录查询时长

- 编写 SQL 语句：

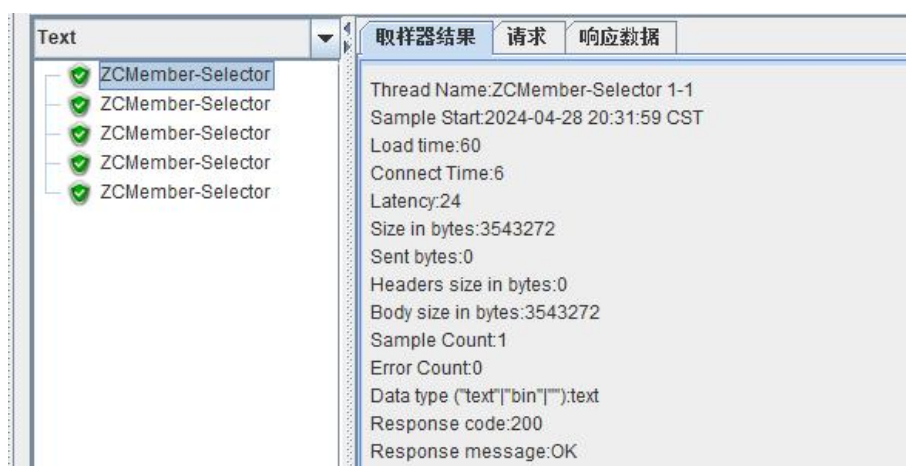


- 测试结果：

第一次到第五次 Load time 分别为：60，41，41，56，38

第一次到第五次 Latency 分别为：24，18，19，26，16

第一次到第五次 Connect Time 分别为：6，0，0，0，0





#### 四、实验结果及分析和（或）源程序调试过程

通过以上对 MySQL 数据库的操作可得到如下结果分析

**1. 对同一表进行 select count(\*) 操作，除第一次的 Load Time 和 Latency 较长之外，其余都接近于 0，Connect Time 第一次稳定在 6-8 之间，其余均为 0**

分析步骤 3-5，猜想对同一数据库中的表进行连续操作，在第一次建立连接之后，接着对其继续操作会保留连接而不释放连接，因此在首次建立连接之后就不需要重新消耗时间建立连接。同样，对同一表进行简单的 select 操作，在 jmeter 本地可能会保存有对应的缓存数据，所以再次对数据库进行相同操作时不需要消耗额外的 Load Time 和 Latency。

**2. 对多个表进行连接查询操作时，Load Time 和 Latency 有下降趋势，但不会像简单的查询一样出现 0 的情况。**

分析步骤 6-7，当进行多表连接操作时，似乎在本地不具有对应的缓存，因为涉及的数据量较大。且表越多，逻辑越复杂的情况下，Load Time 和 Latency 也在增大。但对于 Connect Time 不涉及查询过程，仍具有以上描述特征。

**3. 插入多条记录时，每条记录的时间基本一致**

分析步骤 8，本实验中是每次只插入一条数据反复执行 1000 次，可以发现每次插入的 Load Time 和 Latency 均在 20 左右，Connect Time 仍具有上述特征。

**4. 对数据量不同的表进行 select\* 操作表现出不同的特征**

分析步骤 9-10，步骤 9 中只对 1000 条数据进行查询，可以看到有明显的本地缓存作用，后续时间有明显的降低。但可能因为仍有少部分数据没有缓存到，因此时间也没有降低至 0；而对于数据量为 73818 的 zc\_number 表，缓存引起的作用就不大，每次仍需较大的时间查询数据。

综上，对数据库进行连续操作会有对应的连接保持，而对简单少量数据会有缓存有效降低查询时间，而当数据量较大时，缓存的作用就变小。且查询逻辑越复杂，数据量越大时，对应的 Load Time 和 Latency 也会越大。