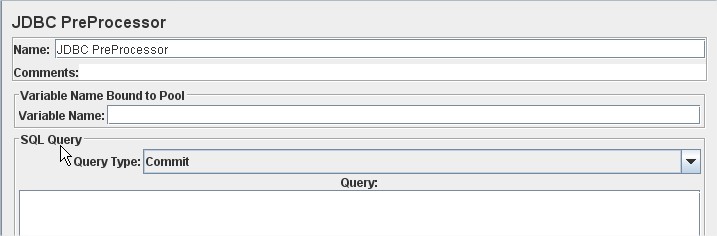
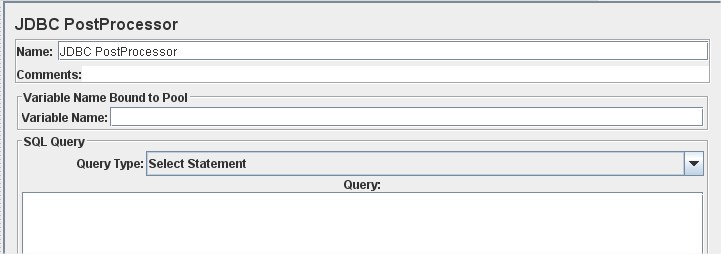
Using JDBC Sampler in JMeter

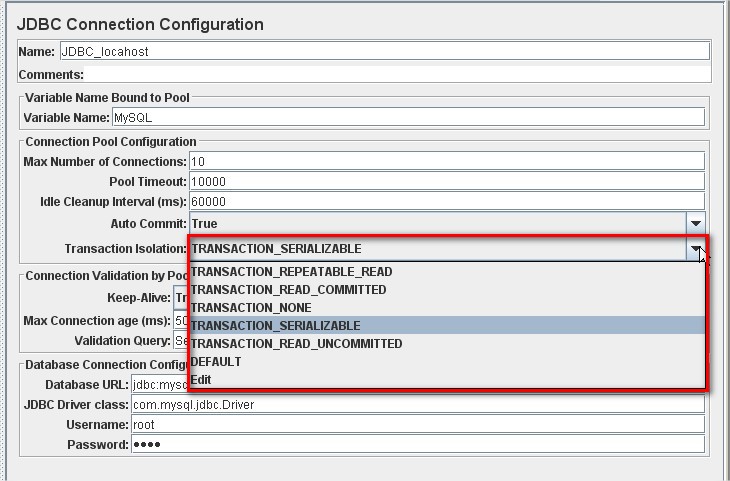
There are 2 JDBC pre- and post-processors (figure 1 and 2),  and “transaction isolation” that is added to JDBC Connection Configuration element (figure 3)



**Figure 1**



**Figure 2**



**Figure 3**

You see that pre- and post-processors are almost the same. Their appointment is to prepare database for testing and to rollback unnecessary changes after testing is done. So, you can specify any SQL query for them to execute. Another thing – “Transaction Isolation” option. And here I should say a few words about SQL transactions.

Transaction…What does it mean? Database transaction is a cycle of SQL queries that are executed against database to search or modify its content. According to definition, a database transaction must be atomic, consistent, isolated and durable. You may know acronym ACID, which describes these properties. Transactions in databases should be isolated from each other to avoid data corruption. Different databases support different isolation levels. Oracle DB that is one of the most common DB’s has 4 levels of isolation: SERIALIZABLE, REPEATABLE READS, READ COMMITTED, READ UNCOMMITTED. For more details you can refer to [Wikipedia - Database Transactions](http://www.google.com/url?q=http%3A%2F%2Fen.wikipedia.org%2Fwiki%2FIsolation_(database_systems)&sa=D&sntz=1&usg=AFQjCNHd5z_1cnuMZ8EF5loa8jiZNdYbsw) . I will just mention that database performance e.g. how much SQL requests it can process per unit time depends on isolation level of transaction, because when transaction has level SERIALIZABLE(the highest one), no another transactions can work with certain data at the same time. And vice verse, multiple READ\_UNCOMMITTED transactions can work with database record simultaneously, but this can cause errors in data if these transactions write some data to DB. There are three types of issues with database transaction:

* Dirty reads (Uncommitted Dependency);
* Non-repeatable reads;
* Phantom reads;

A dirty read happens if a transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed. See example in below.

**Transaction 1** **Transaction 2**

|  |  |
| --- | --- |
| /\* Query 1 \*/ SELECT age FROM users WHERE id = 1; /\* will read 20 \*/ |  |
|  | /\* Query 2 \*/UPDATE users SET age = 21 WHERE id = 1;/\* No commit here \*/ |
| /\* Query 1 \*/ SELECT age FROM users WHERE id = 1; /\* will read 21 \*/ |  |
|  | ROLLBACK; /\* lock-based DIRTY READ \*/ |

A non-repeatable read occurs, when during the course of a transaction, a row is retrieved twice and the values within the row differ between reads.

**Transaction 1** **Transaction 2**

|  |  |
| --- | --- |
| /\* Query 1 \*/ SELECT \* FROM users WHERE id = 1; |  |
|  | /\* Query 2 \*/ UPDATE users SET age = 21 WHERE id = 1; COMMIT; /\* in multiversion concurrency control, or lock-based READ COMMITTED \*/ |
| /\* Query 1 \*/ SELECT age FROM users WHERE id = 1; /\* will read 21 \*/ |  |
|  | /\* Query 1 \*/ SELECT \* FROM users WHERE id = 1; COMMIT; /\* lock-based REPEATABLE READ \*/ |

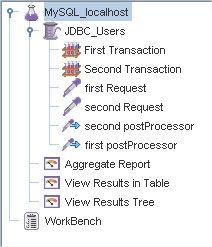
A phantom read occurs when, in the course of a transaction, two identical queries are executed, and the collection of rows returned by the second query is different from the first.

**Transaction 1** **Transaction 2**

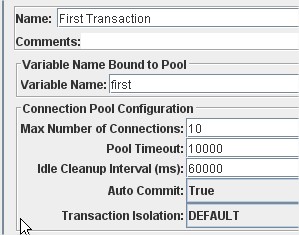
|  |  |
| --- | --- |
| /\* Query 1 \*/ SELECT \* FROM usersWHERE age BETWEEN 10 AND 30; |  |
|  | /\* Query 2 \*/ INSERT INTO users VALUES ( 3, 'Bob', 27 ); COMMIT; |
| /\* Query 1 \*/ SELECT \* FROM users WHERE age BETWEEN 10 AND 30; |  |

So, it’s very important for developer to choose appropriate isolation level for SQL queries. And that’s why performance testers need to take into consideration this parameter while developing test-plan.

yOU can create SQL queries of any configurations. Let’s see, how it works.



**Figure 4**

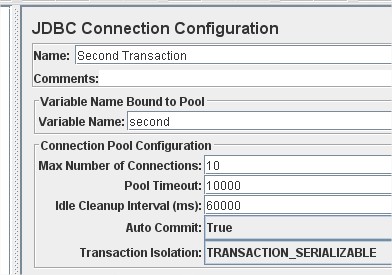


**Figure 5**

For my testing I used local MySQL database that is installed on my desktop

First, I add one JDBC Connection Configuration. I fill “Database connection configuration” section with valid credentials to database and set name “First Transaction” for this connection. Note, that I set “Transaction Isolation” to TRANSACTION\_READ\_UNCOMMITED that means that another transaction can have read/write access to data, which is used by current transaction. “first” transaction will contain only one JDBC request with the following SQL query: “UPDATE world.city SET population=1000000 WHERE Name like '%'”

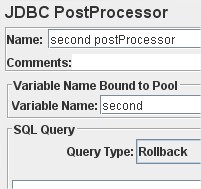
Then I create “Second Transaction” JDBC Connection Configuration. The only its difference from first transaction is that I set “TRANSACTION\_SERIALIZABLE” as transaction isolation option (figure 6).



**Figure 6**

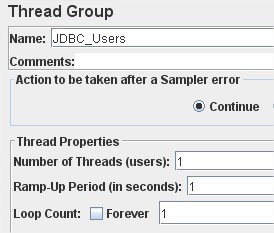
“Second Transaction” contains the same JDBC Request.

Both JDBC PostProcessors contain query “Rollback” to revert all changes in database to latest stable state(figure 7).



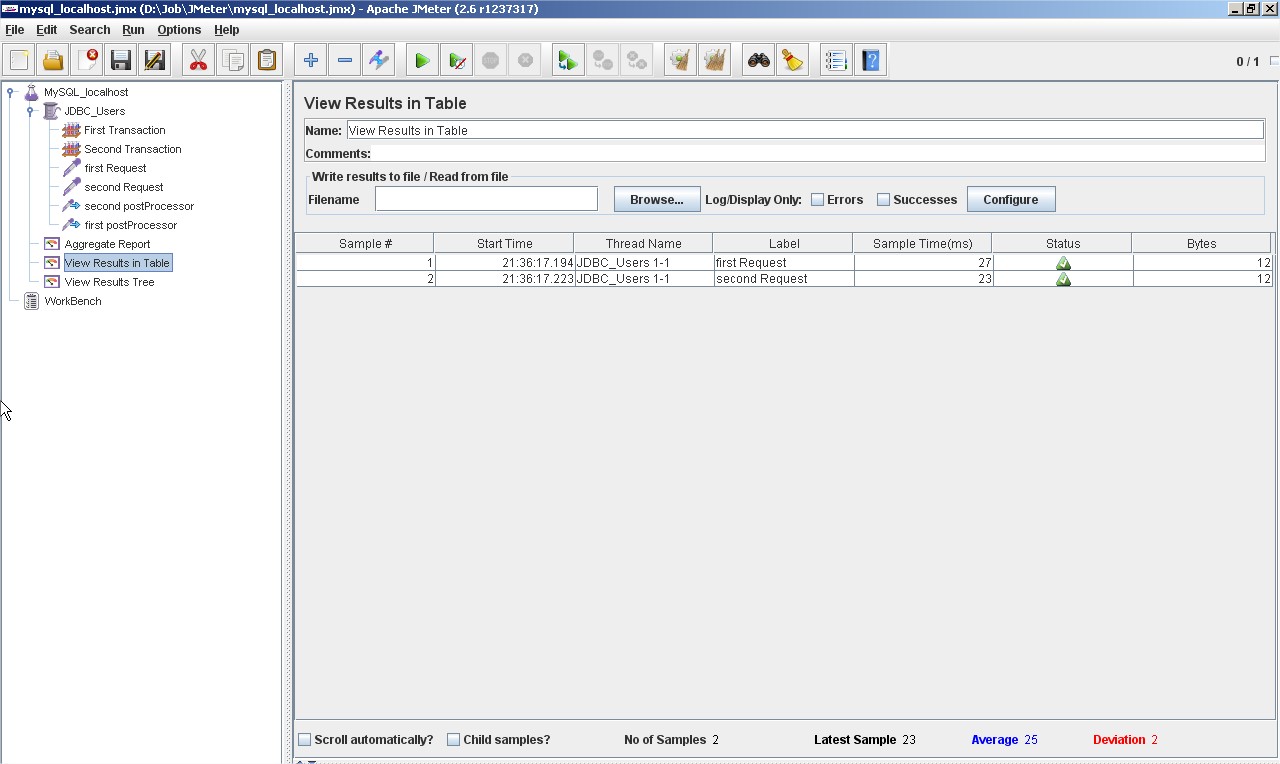
**Figure 7**

I’ll set only one user for “JDBC\_Users” Thread group (see figure 8).



**Figure 8**

OK, I press “Start” button. Test is executed and what I see in “View Results in Table” listener (figure 9)?



**Figure 9**

Despite of the fact that connection 1 was created earlier, second request has less execution time.

It happened because second transaction had SERIALIZABLE level and first only READ\_UNCOMMITTED. So 2nd occupied database table until it ended and only after that first request could complete its job.

So, you see that database performance depends on transaction isolation levels. That’s why opportunity of selecting isolation level of JDBC request is important for database performance testing.