Assignment 2 : Transactions in SQL

Hugo BEHERAY - Clément FAGES

Assignment 2 BDD - 19/10/2021

Preamble

To complete this lab we have installed MySQL Workbench instead of *WampServer* to launch queries and have more options for our work. For example we can disable the *autocommit* on Workbench which doesn't exist on *WampServer*. Now we can start our work by writing 'start transaction' in order to do commit or rollback. We have imported the script *'company.sql'* from campus on Workbench to get some data (database, tables, tuples..).

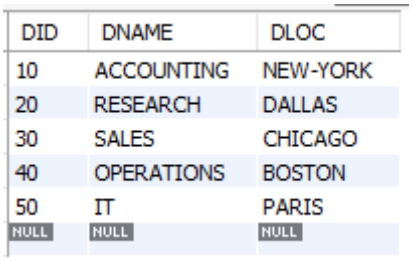
Tasks

1. **Commit and rollback**

   The transaction should look like that :

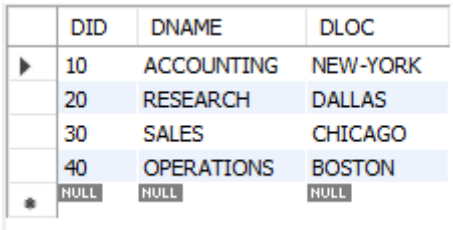
```
start transaction;
insert into *dept* values ('50', 'IT', 'PARIS');
select * from dept;
rollback;
select * from dept;
```

   First, we start a new transaction by writing *'start transaction*'. Then we insert a new value in the *dept* table. Next we select all the elements in the table *dept* and we have the following result :



DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
NULL	NULL	NULL

   Let's do a *rollback* and re-select everything to see what happened (line 4 and 5) :



	DID	DNAME	DLOC
▶	10	ACCOUNTING	NEW-YORK
	20	RESEARCH	DALLAS
	30	SALES	CHICAGO
	40	OPERATIONS	BOSTON
*	NULL	NULL	NULL

   The table came back as it was. So it canceled the modifications. Here we try the same query with *commit* instead rollback :

```
start transaction;
insert into dept values ('50', 'IT', 'PARIS');
select * from dept;
commit;
select * from dept;
```

Here is the result :

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
NULL	NULL	NULL

The insert has been saved, there is no difference between the two tables of the commits.

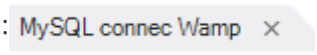
2. **Client failure**

The new query is the following :

```
start transaction;
insert into dept values ('60', 'PRODUCT', 'BERLIN');
select * from dept;
```

We started a new transaction and insert new *dept*. Then we select the elements from the table and we have this :

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
60	PRODUCT	BERLIN
NULL	NULL	NULL

Now we close the tab by clicking here : 

Then we close MySQL to launch it again. We enter the following query again :

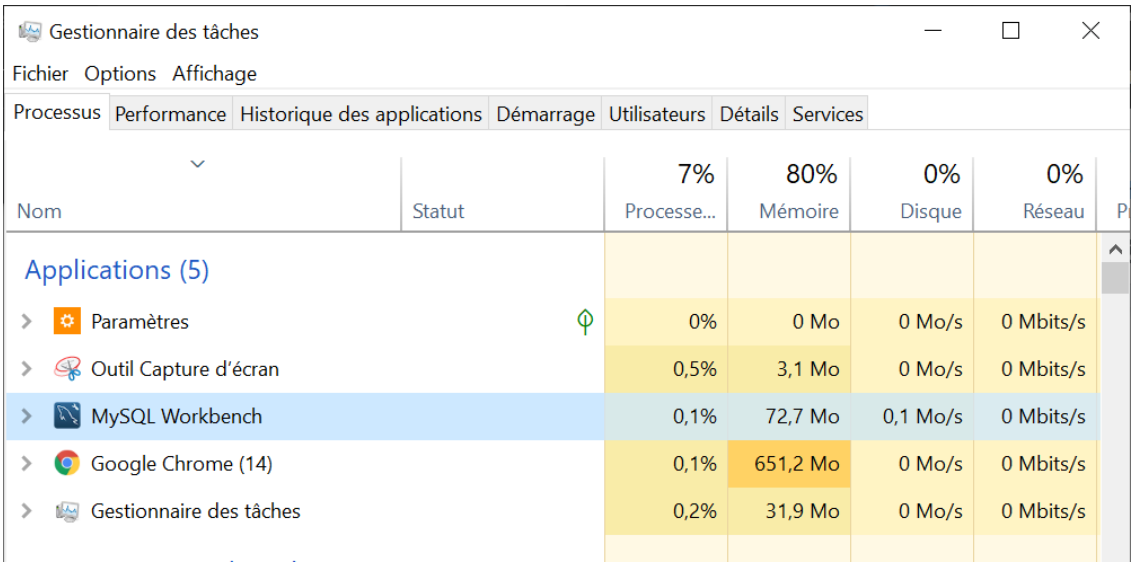
```
select * from dept;
```

We get this result :

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
NULL	NULL	NULL

The insertion has not been saved because it is not in the table. As we didn't commit it, the information modified have not been saved.

Now we will try to repeat the operations but we will abruptly stop the process with the task manager.



We do a right click on the MySQL Workbench and stop the process. Now we re-open it and we select the tuples in the *dept* table.

Here is the result :

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
NULL	NULL	NULL

It didn't save the transaction.
So we conclude that if we don't commit, modifications will not be saved.

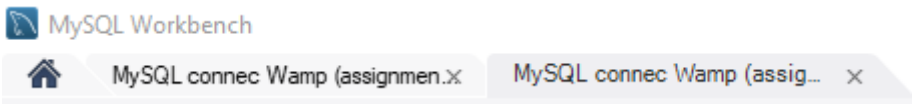
3. **Transaction Isolation**

To get the default transaction isolation level of the database we have to enter the following command :

```
'show variables like '%isolation%''
```

The value displayed is : 'REPEATABLE-READ'.
It means that if we do a read operation, it will use the snapshot information established by the first read to present query results based on a point in time regardless of changes performed by other transactions running at the same time.

To do the experiment, we will create another connection to the database so we have two tabs :



We will start two transactions and do operations on the exact same table from the same database.

In the first connection we have this code :

```
start transaction;
insert into dept values ('60', 'PRODUCT', 'BERLIN');
select * from dept;
```

We launch it and we get this result :

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
60	PRODUCT	BERLIN
NULL	NULL	NULL

The new tuple has been well inserted.
In the other tab we also create a new transaction and we select the table *dept* from the database :

```
start transaction;
select * from dept;
```

Here is the result :

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
NULL	NULL	NULL

The previous insertion is not displayed because it is not part of this transaction.

So the modifications a transaction makes to the database are only visible to that transaction.

Now we will repeat the experiment by deleting the department inserted previously.

Let's admit that we have the same database in the two transactions so that we can see if we delete a tuple in one, the other tuple will notice it. In the first connection we have now this code :

```
start transaction;
delete from dept where DID = 60;
select * from dept;
```

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
NULL	NULL	NULL

We can see that the has been deleted the tuple.

In the other tab we just select all the information from *dept* and we have this result :

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
60	PRODUCT	BERLIN
NULL	NULL	NULL

So the row has not been deleted in the second transaction via the second connection.

4. **Isolation levels**

We set the transaction level to read uncommitted thanks to this line of code :

```
set session transaction isolation level READ UNCOMMITTED;
show variables like '%isolation%';
```

We get this output :

Variable_name	Value
transaction_isolation	READ-UNCOMMITTED

We will repeat the previous experiment with this value.

First, we will add a new tuple in the table *dept* in *WampServer* and refresh the database on Workbench. So then, we will delete it in one connection to see if the transaction in the second connection noticed it.
Here is our first table :

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
60	PRODUCT	BERLIN
NULL	NULL	NULL

Now we delete the last row :

```
start transaction;
delete from dept where DID = 60;
select * from dept;
```

Here is our new table :

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
NULL	NULL	NULL

Now in the other connection tab, we will select the elements of the table and see if the row has been deleted.

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	
30	SALES	
40	OPERATIONS	BOSTON
50	IT	PARIS
NULL	NULL	NULL

The row has been deleted.
The option *read uncommitted* allows that transaction may see uncommitted changes made by some other transaction.
To modify the behavior from the previous exercise, we only have to edit one transaction. The transaction that must be edited is the one where we don't delete the row and we just select the information from the table. Indeed it will be able to get elements from uncommitted transactions. However, we can change the isolation level in both tabs.

5. **Isolation levels - Continued**

Let's set the isolation level to serializable :

```
set session transaction isolation level serializable;
```

This time we will add the row *('50', 'IT', 'PARIS')* in the database :

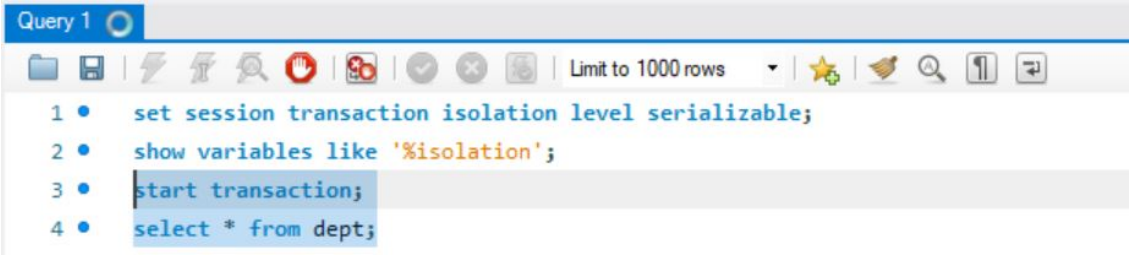
```
start transaction;
insert into dept values ('50', 'IT', 'PARIS');
select * from dept;
```

Now we display the first table :

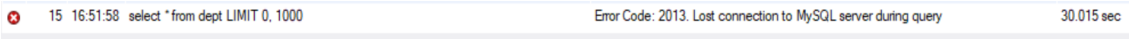
DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
NULL	NULL	NULL

In the second connection we try to display the table by starting a new transaction :

The query is running for a long time and nothing is displayed. The following screenshot is what I have on my screen :



The query is running. We just try to select all the elements from *dept*.



At the end of the query we got an error.

Indeed, serializable requires read and write locks to be released at the end of the transaction. So we have to commit our first transaction from the first connection to get the values from the tab *dept* from the second transaction.

If we commit our first transaction we get a result this time for the second transaction :

DID	DNAME	DLOC
10	ACCOUNTING	NEW-YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	IT	PARIS
NULL	NULL	NULL

Exactly the same as the output from the first transaction.

6. **JBDC**

To complete this part, we edited the code in *Test.java* and *DataAccess.java*.

To get the autocommit value we use this function : *getAutoCommit();*

To get the isolation level we use : *getTransactionIsolation();*

Then we added a try, catch condition around the statements and prepared statements. So if there is an error, we can rollback or commit at the end if everything was ok.