# Assignement 2 – advanced database

**Jacques Polart – inge4 – groupe 4**

## 1 ) Commit and rollback :

```
start transaction;
insert into DEPT values (50, 'test', 'test');
select * from DEPT;
rollback;
select * from DEPT;
```

on the first select we can see the added line, after the rollback, the added line is no more there.

```
start transaction;
insert into DEPT values (50, 'test', 'test');
select * from DEPT;
commit;
select * from DEPT;
```

we still can see the added line after the commit.

## 2) Client failure :

if the program is closed abruptly, the line isn't added, else the transaction works fine.
Conclusion : all changes not commited will be abandoned.

## 3) TransactionIsolation :

```
show variables like '%isolation%'
```
→ 'transaction_isolation', 'REPEATABLE-READ'

```
start transaction;
insert into DEPT values (50, 'test1', 'test1');
start transaction;
insert into DEPT values (60, 'test2', 'test2');
select * from DEPT;
select * from DEPT;
commit;
commit;
```

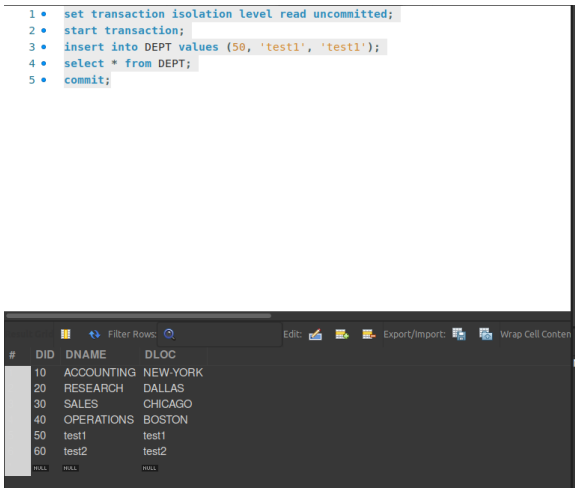
before the commits, we cant see changes from other transaction, only thoses from the current transaction.

## 4) Isolation levels :

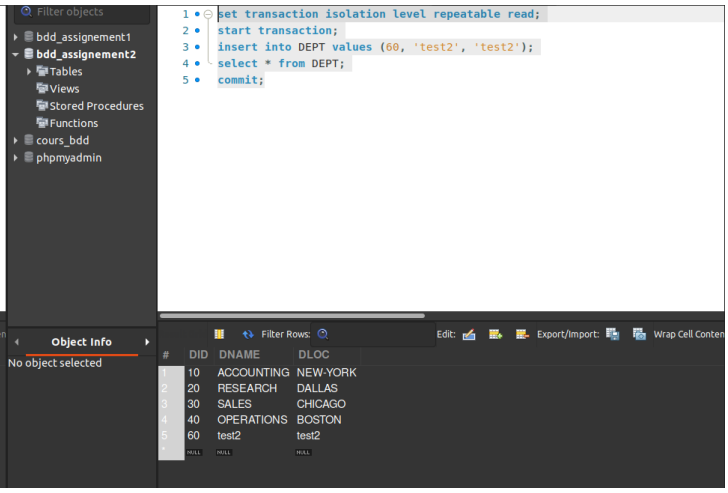
```
set transaction isolation level read uncommitted;
start transaction;
insert into DEPT values (50, 'test1', 'test1');
set transaction isolation level repeatable read;
start transaction;
insert into DEPT values (60, 'test2', 'test2');
select * from DEPT;
select * from DEPT;
commit;
commit;
```

the transaction that has the ‘READ UNCOMMITED’ attribte will be able to see changes from other transactions.
Here, Transaction 1 see changes of both transactions, when transaction 2 see only it’s own changes.

Transaction 1




Transaction 2

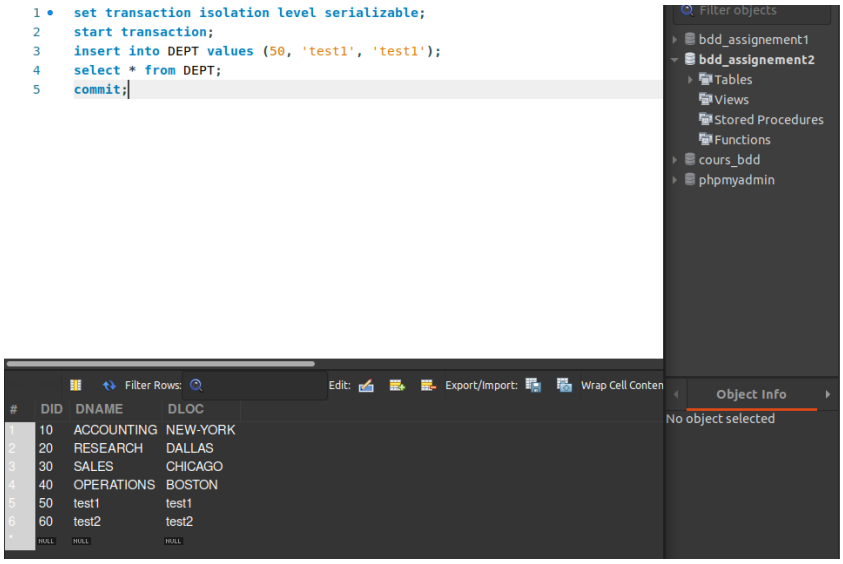


## 5) Isolation levels – Continued :

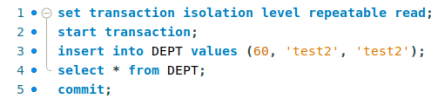
```
set transaction isolation level serializable;
start transaction;
insert into DEPT values (50, 'test1', 'test1');
set transaction isolation level repeatable read;
start transaction;
insert into DEPT values (60, 'test2', 'test2');
select * from DEPT;
select * from DEPT;
commit;
commit;
```

In this example, the select from transaction 1 await the commit of the transaction 2, to execute.
So transaction 1 can also see changes made by transaction 2.

Transaction 1




transaction 2




## 6) JDBC code :

Solution given isn't working due to version compatibility problems.
The code i wrote has not been tested so. But i hope it suit the logic i have in mind.