

Assigment 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 committed 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 UNCOMMITTED' attribute will be able to see changes from other transactions.

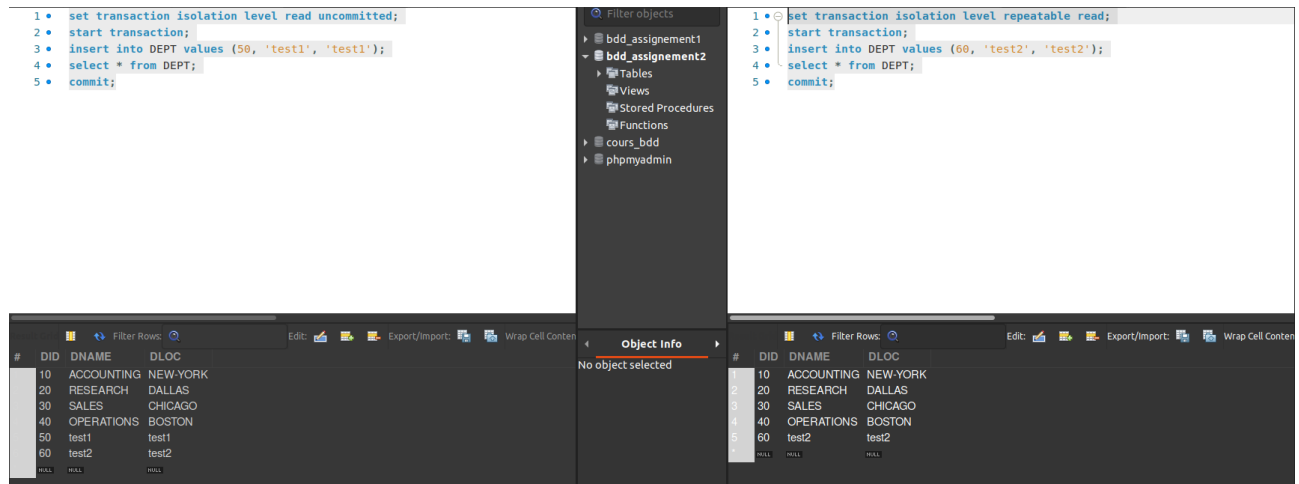
Here, **Transaction 1** see changes of both transactions, when **transaction 2** see only it's own changes.

Transaction 1

```
1 • set transaction isolation level read uncommitted;  
2 • start transaction;  
3 • insert into DEPT values (50, 'test1', 'test1');  
4 • select * from DEPT;  
5 • commit;
```

Transaction 2

```
1 • set transaction isolation level repeatable read;  
2 • start transaction;  
3 • insert into DEPT values (60, 'test2', 'test2');  
4 • select * from DEPT;  
5 • commit;
```



#	DID	DNAME	DLOC
1	10	ACCOUNTING	NEW-YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON
5	50	test1	test1
6	60	test2	test2

#	DID	DNAME	DLOC
1	10	ACCOUNTING	NEW-YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON
5	60	test2	test2

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

```
1 • set transaction isolation level serializable;
2 • start transaction;
3 • insert into DEPT values (50, 'test1', 'test1');
4 • select * from DEPT;
5 • commit;
```

transaction 2

```
1 • set transaction isolation level repeatable read;
2 • start transaction;
3 • insert into DEPT values (60, 'test2', 'test2');
4 • select * from DEPT;
5 • commit;
```

The screenshot shows a database management tool interface. On the left, a SQL editor displays two transactions. Transaction 1 (Transaction 1) is highlighted in blue and contains the following SQL statements: `set transaction isolation level serializable;`, `start transaction;`, `insert into DEPT values (50, 'test1', 'test1');`, `select * from DEPT;`, and `commit;`. Transaction 2 (transaction 2) is highlighted in orange and contains the following SQL statements: `set transaction isolation level repeatable read;`, `start transaction;`, `insert into DEPT values (60, 'test2', 'test2');`, `select * from DEPT;`, and `commit;`. On the right, a sidebar shows a tree view of database objects, including `bdd_assignment1`, `bdd_assignment2`, `Tables`, `Views`, `Stored Procedures`, `Functions`, `cours_bdd`, and `phpmyadmin`. At the bottom, a table view displays the contents of the `DEPT` table. The table has columns `DID`, `DNAME`, and `DLOC`. The data rows are: `10 ACCOUNTING NEW-YORK`, `20 RESEARCH DALLAS`, `30 SALES CHICAGO`, `40 OPERATIONS BOSTON`, `50 test1 test1`, and `60 test2 test2`.

#	DID	DNAME	DLOC
1	10	ACCOUNTING	NEW-YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON
5	50	test1	test1
6	60	test2	test2

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.