LABORATORY REPORT

LABORATORY REPORT

Common Data	
Student name	**Kormoua Khongmeng**
Neptun code	**I3MLPQ**
Department	**Dept. of Automation and Applied Informatics**
Instructor name	**AL-Magsoosi Husam Kareem Farhan**
Laboratory place	**BME IL206**
Laboratory time	**10:15 – 12:00**
Title or Sequence number	**4**

Exercises	
Task 1	☒
Task 2	☒
Task 3	☒
Task 4	☒
Task 5	☒
Task 6	☒
Task 7	☒
Task 8	☒
Task 9	☒
Task 10	☒
Task 11	☒
Task 12	☒
Task 13	☒

© BME – AAIT 2015

EXERCISES

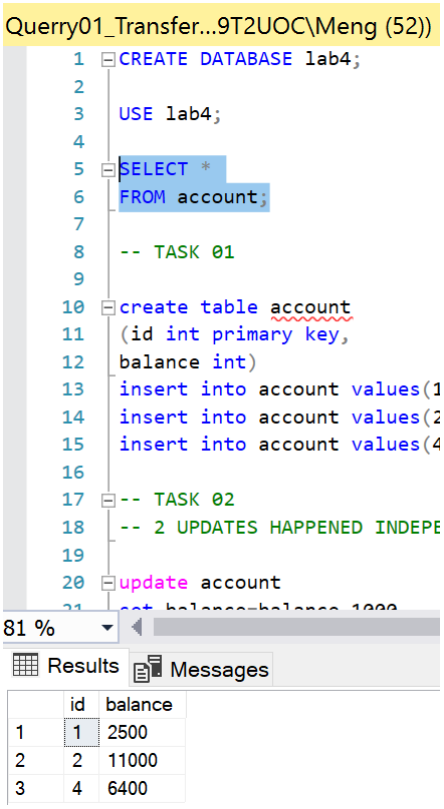
TASK #1

Problem statement: Get connected to the SQL Server and create a bank account database using the following SQL script.

Solution:

```
create table account
(id int primary key,
balance int)
insert into account values(1, 5000)
insert into account values(2, 8500)
insert into account values(4, 6400);
```

Visual result:



Reasoning:

I just created a table as the guide said.

© BME – AAIT 2015

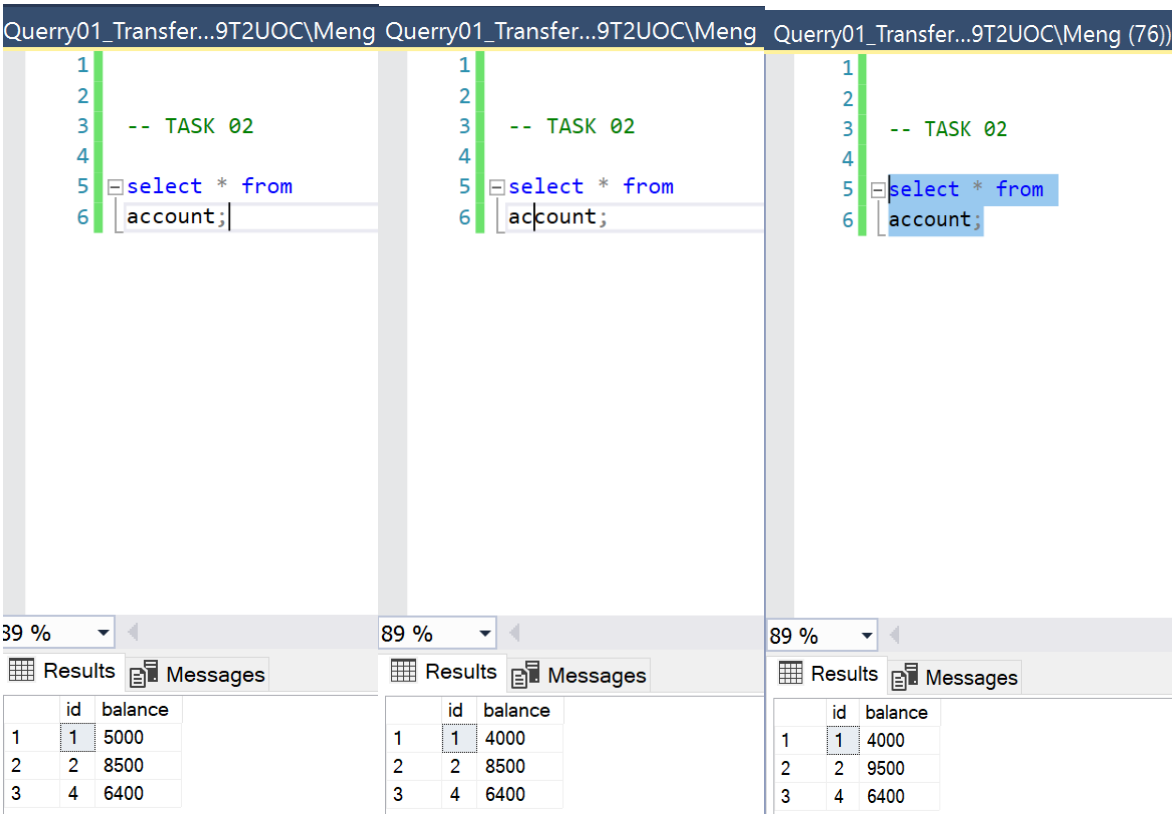
TASK #2

Problem statement: Transfer 1000 HUF from bank account 1 to 2. In order to check the transfer open another query window, this creates a second parallel connection to the database. Check balance during transfer continuously. What did you notice and why?

Solution:

```
select * from
account;
update account
set balance=balance-1000
where id=1;
select * from
account;
update account
set balance=balance+1000
where id=2;
select * from
account;
```

Visual result:



Before transaction | At the middle of transaction | After the transaction

Reasoning:

We can see that this is not a single process or atomic process. they are a state that 1000 balance just disappear from the database which is because I did not begin transaction and commit as atomic manner.

© BME – AAIT 2015

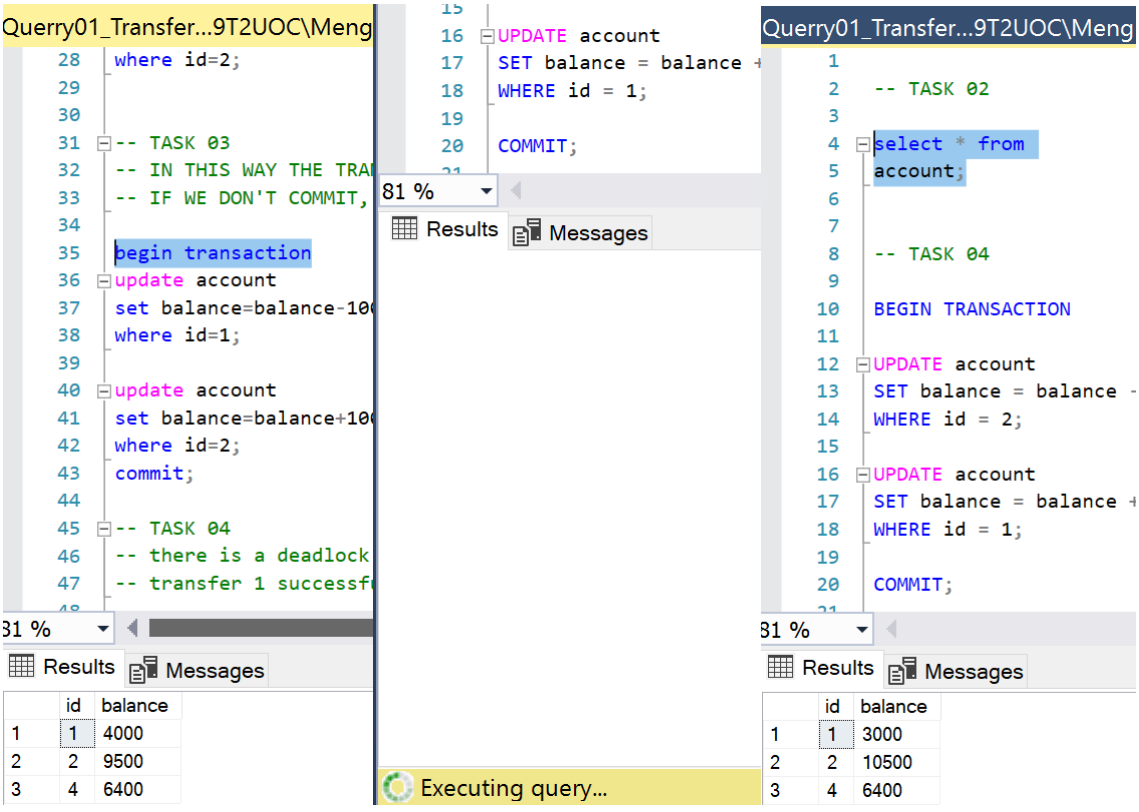
TASK #3

Problem statement: Change the above script in a way that the data manipulation operations are put into the same transaction. Replay the second task. What did you notice and why?

Solution:

```
begin transaction
select * from account;
update account
set balance=balance-1000
where id=1;
select * from account;
update account
set balance=balance+1000
where id=2;
select * from account;
commit;
select * from account;
```

Visual result:



Before an update | Between the update | After the commit

Reasoning:

Here before we transfer balance, we begin transaction and after that we commit transaction. This will make our process an atomic process. on the other script which I use as a second person that want to check the balance of the account; we see that during the transaction the second person has to wait until the first transaction is done since SSMS use “read committed” isolation level as default.

© BME – AAIT 2015

TASK #4

Problem statement: Deadlock creation. There are two transfers running in parallel: one of them sends 500 HUF from account 1 to 2 while the second one sends 300 HUF from bank account 2 to 1. Let the transactions schedule be the following:

Solution:

First transaction	Second transaction
`BEGIN TRANSACTION`	
	`BEGIN TRANSACTION`

`UPDATE account`
`SET balance = balance - 500`
`WHERE id = 1;`	

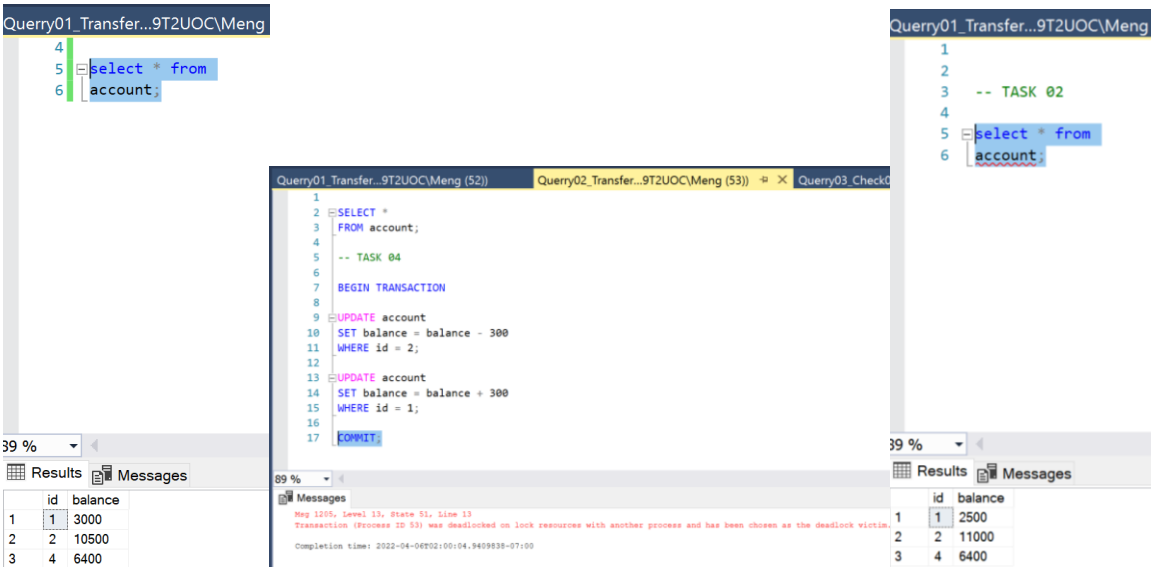
	`UPDATE account`
	`SET balance = balance - 300`
	`WHERE id = 2;`

`UPDATE account`
`SET balance = balance + 500`
`WHERE id = 2;`	

	`UPDATE account`
	`SET balance = balance + 300`
	`WHERE id = 1;`

`COMMIT;`	

Visual result:



Before transaction | Transaction 2 is aborted due to deadlock | After transaction

Reasoning:

We see that a deadlock happened since both tried to access the same resource at the same time. One of the transactions is forced to be aborted and only one of the transactions is executed successfully which in our case is the first transaction.

© BME – AAIT 2015

TASK #5

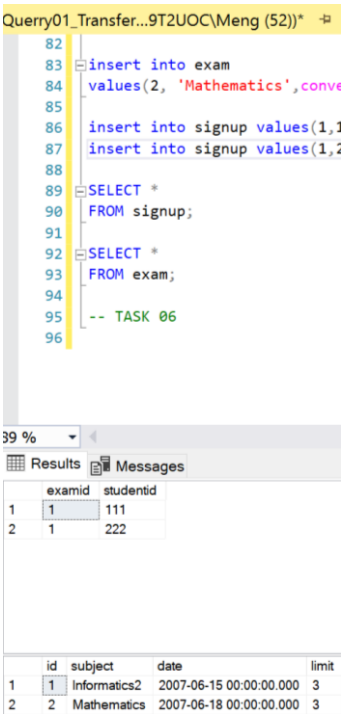
Problem statement: Create a simple exam signup system using the following SQL script.

Solution:

```
drop table signup;
drop table exam;

create table exam
( id int primary key,
 subject varchar(20),
 date datetime,
 limit int
);
create table signup(
 examid int references exam(id),
 studentid int,
 primary key (examid,studentid)
);
insert into exam
values(1, 'Informatics2',convert(datetime,'2007.06.15',102),3);
insert into exam
values(2, 'Mathematics',convert(datetime,'2007.06.18',102),3);
insert into signup values(1,111);
insert into signup values(1,222);
```

Visual result:



Reasoning:

Here I just created a table based on the given script.

© BME – AAIT 2015

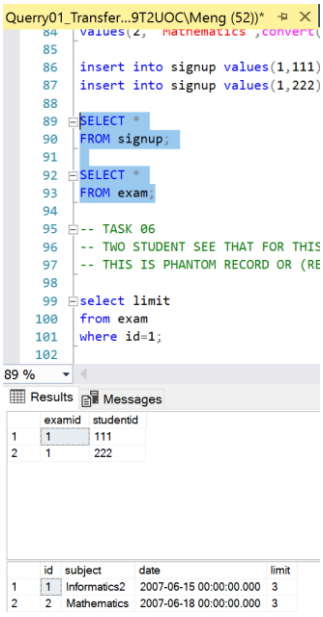
TASK #6

Problem statement: Simulate two concurrent signups to the first exam. Schedule of the processes should be the following:

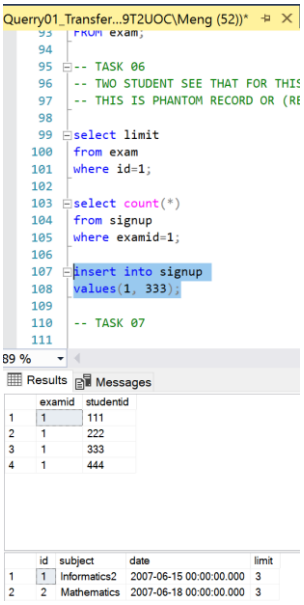
Solution:

Step	Student 1's signup	Student 2's signup
1	`select limit from exam where id=1;`	
2		`select limit from exam where id=1;`
3	`select count(*) from signup where examid=1;`	
4		`select count(*) from signup where examid=1;`
5	`insert into signup values(1, 333);`	
6		`insert into signup values(1, 444);`

Visual result:



State before transactions



State after transactions

Reasoning:

We have a problem that two transactions are not properly isolated (phantom read). In this case both student wants to register to a course, two students check the available place, and two students see that there is 1 available place, so both of them register at the same time, since this is not done in atomic manner with some proper isolation level (default level read committed cannot prevent this) then both get registered in the exam.

© BME – AAIT 2015

TASK #7

Problem statement: If transactions are isolated properly, the problem described in exercise 6 can be avoided. Increase isolation levels of transactions to 'serializable' and replay the previous task as follows.

Solution:

Step	First signup	Second signup
1	`set transaction isolation level serializable;` `Begin transaction`	
2		`set transaction isolation level serializable;` `Begin transaction`

3	`select limit`
	`from exam`
	`where id=1;`	

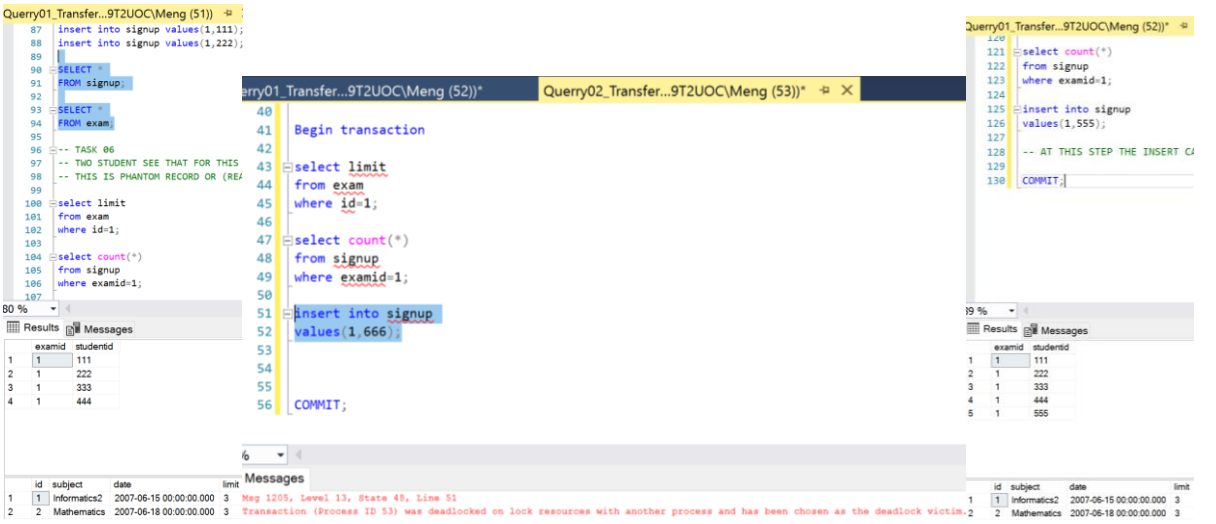
4		`select limit`
		`from exam`
		`where id=1;`

5	`select count(*)`
	`from signup`
	`where examid=1;`	

6		`select count(*)`
		`from signup`
		`where examid=1;`

7	`insert into signup` `values(1,555);`	
		`insert into signup` `values(1,666);`

Visual result:



State before transactions deadlock happen State after transactions

Reasoning:

In this serializable isolation level, we can solve problem about phantom read, by while two transactions try to insert value into table, a deadlock happened, one of the transactions is aborted and the other one is successfully executed.

© BME – AAIT 2015

TASK #8

Problem statement: Increase limit of the first exam to 5. Replay example 7. What is the result? Why?

Solution:

Step	First signup	Second signup

1	`UPDATE exam`
	`SET limit = 5`
	`WHERE exam.id = 1;`	

2	`set transaction isolation level serializable;` `Begin transaction`	
3		`set transaction isolation level serializable;` `Begin transaction`

4	`select limit`
	`from exam`
	`where id=1;`	

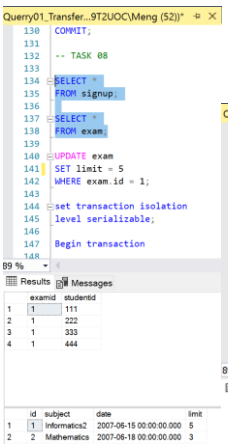
5		`select limit`
		`from exam`
		`where id=1;`

6	`select count(*)`
	`from signup`
	`where examid=1;`	

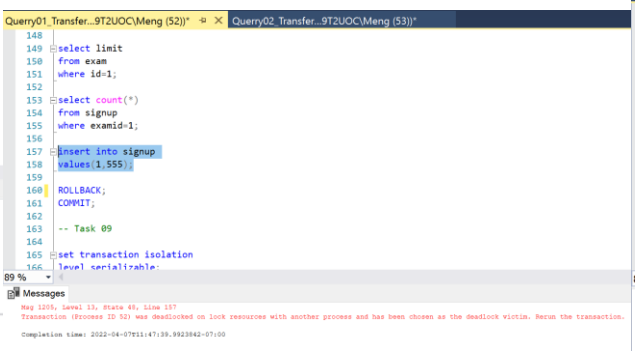
7		`select count(*)`
		`from signup`
		`where examid=1;`

8	`insert into signup values(1,555);`	
9		`insert into signup values(1,666);`

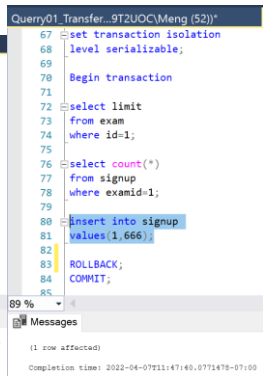
Visual result:



State before transactions



deadlock happen



State after transaction

Reasoning:

Here we increase the limit of the first exam to 5 and repeat steps from task 7, both will see that there is enough space for 2 people. However, if they tried to insert at the same time, a deadlock happened again and only one of them is registered to the exam.

© BME – AAIT 2015

TASK #9

Problem statement: Replay example 7 but the first student should signup to the first exam while the second one should signup to the second exam. What is the result? Why?

Solution:

Step	First signup	Second signup
1	`set transaction isolation level serializable;` `Begin transaction`	
2		`set transaction isolation level serializable;` `Begin transaction`

3	`select limit`
	`from exam`
	`where id=1;`	

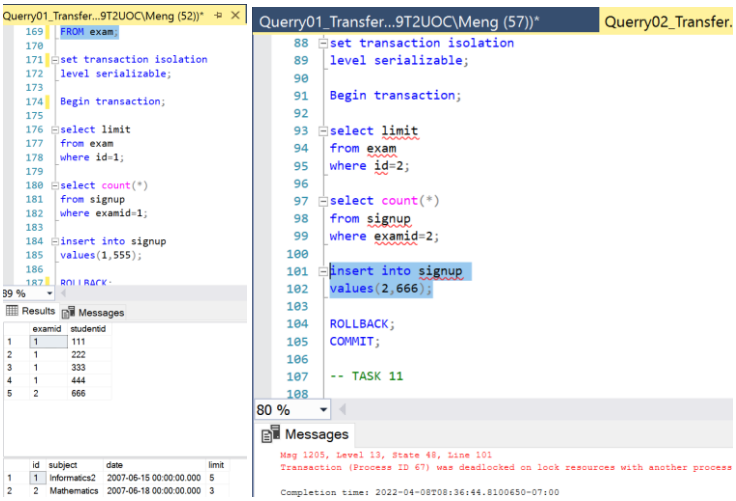
4		`select limit`
		`from exam`
		`where id=2;`

5	`select count(*)`
	`from signup`
	`where examid=1;`	

6		`select count(*)`
		`from signup`
		`where examid=2;`

7	`insert into signup` `values(1,555);`	
8		`insert into signup` `values(2,666);`

Visual result:



Reasoning:

In this "serialization" isolation level, even though, two students tried to read and register to a different exam, but only one of their transactions will be executed successfully while the other one will be aborted as previous task.

© BME – AAIT 2015

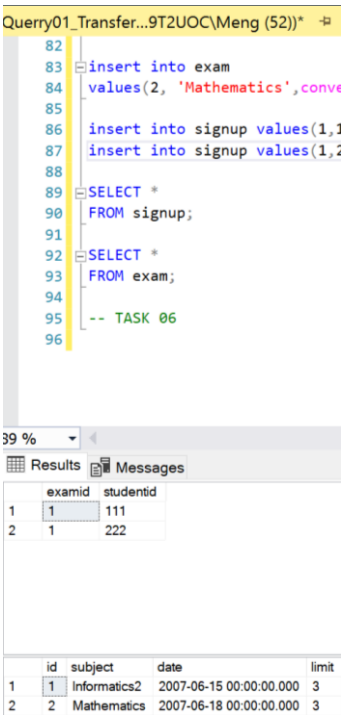
TASK #10

Problem statement: Reset our database by executing the script of example 5 again

Solution:

```
drop table signup;
drop table exam;
create table exam
( id int primary key,
 subject varchar(20),
 date datetime,
 limit int
);
create table signup(
 examid int references exam(id),
 studentid int,
 primary key (examid,studentid)
);
insert into exam
values(1, 'Informatics2',convert(datetime,'2007.06.15',102),3);
insert into exam
values(2, 'Mathematics',convert(datetime,'2007.06.18',102),3);
insert into signup values(1,111);
insert into signup values(1,222);
```

Visual result:



Reasoning:

Here I just drop and recreate tables based on the given code.

© BME – AAIT 2015

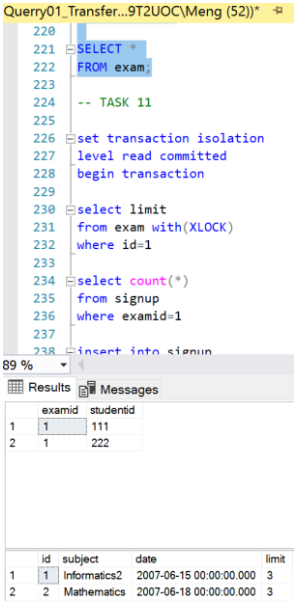
TASK #11

Problem statement: The efficiency of the system can be increased by using read committed isolation level and mutual locking. If concurrent transactions lock only the record they need, mutual exclusion can be achieved. The schedule of the processes should be the following:

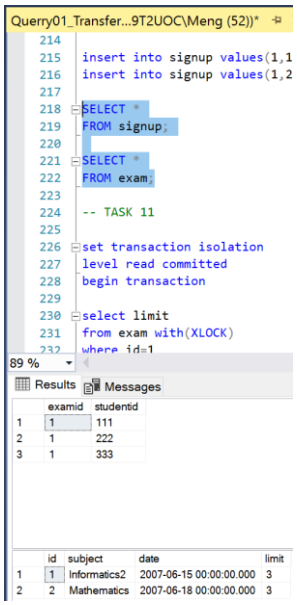
Solution:

Step	First signup	Second signup
1	`set transaction isolation level read committed begin transaction`	
2		`set transaction isolation level read committed begin transaction`
3	`select limit from exam with(XLOCK) where id=1`	
4		`select limit from exam with(XLOCK) where id=1`
5	`select count(*) from signup where examid=1`	
6	`insert into signup values(1,333)`	
7	`commit`	
8		`select count(*) from signup where examid=1`
9		`commit`

Visual result:



State before transaction



State after transacion

© BME – AAIT 2015

Reasoning:

In this approach we use “read committed” isolation level which solve a problem that 2 students see an available place for them if they check it at the same time. In this approach, if one already started the transaction, the other one cannot even read anything from the table until the first student commit his transaction. Not after the first student committed his transaction, the second student see that no place is available for him. In this way 2 of them will not be able to register and go exceed the limit number of student of the exam.

© BME – AAIT 2015

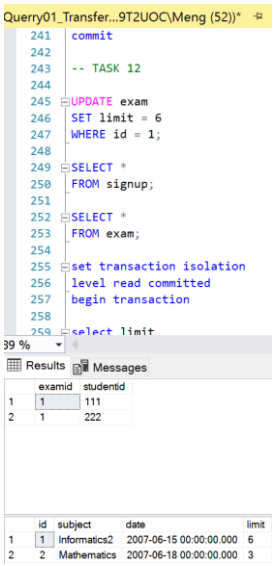
TASK #12

Problem statement: Increase limit of the first exam to 6. Replay example 11. What is the result? Why?

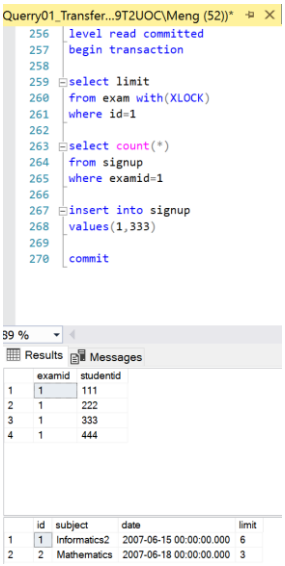
Solution:

Step	First signup	Second signup
1	`UPDATE exam SET limit = 6 WHERE id = 1;`	
2	`set transaction isolation level read committed begin transaction`	
3		`set transaction isolation level read committed begin transaction`
4	`select limit from exam with(XLOCK) where id=1`	
5		`select limit from exam with(XLOCK) where id=1`
6	`select count(*) from signup where examid=1`	
7	`insert into signup values(1,333)`	
8	`commit`	
9		`select count(*) from signup where examid=1`
10		`insert into signup values(1,444);`
11		`commit`

Visual result:



State before transactions



State after transaction

© BME – AAIT 2015

Reasoning:

Here we set the limit to the number where both student can be registered in to the exam. Again with this “read committed” isolation level, if both start the transaction, one of them cannot even read anything until the other commit their transaction. After the first student committed his transaction, the second one now can read and check the available place for him. He will see that he still can register and he will get registered since the limit has been increased and there is still a place for him.

© BME – AAIT 2015

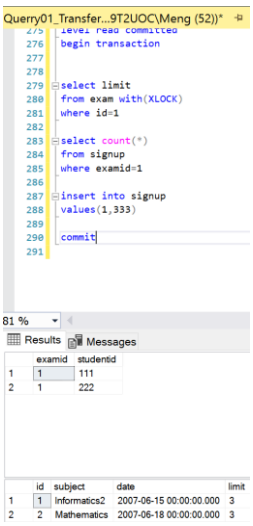
TASK #13

Problem statement: Replay example 12 but the first student should signup to the first exam while the second one should signup to the second exam. What is the result? Why?

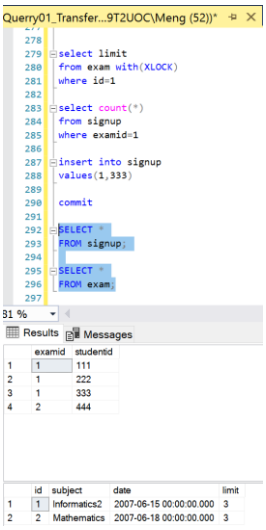
Solution:

Step	First signup	Second signup
1	`UPDATE exam SET limit = 6 WHERE id = 1;`	
2	`set transaction isolation level read committed begin transaction`	
3		`set transaction isolation level read committed begin transaction`
4	`select limit from exam with(XLOCK) where id=1`	
5		`select limit from exam with(XLOCK) where id=2`
6	`select count(*) from signup where examid=1`	
7	`insert into signup values(1,333)`	
8	`commit`	
9		`select count(*) from signup where examid=2`
10		`insert into signup values(2,444);`
11		`commit`

Visual result:



State before transactions



State after transaction

© BME – AAIT 2015

Reasoning:

For this approach “read committed” isolation level, if both wants to access the same resource, they need to wait for the other one to commit there transaction first. But in this case, two of them tried to register for different exam. They can read, access and insert into table independently from each other without any waiting at all. Everything is perfect in this case.

© BME – AAIT 2015

INSTRUCTIONS

1. **Problem statement is mandatory.**
2. **A solution without explanation is NOT accepted.**
3. **If you need to copy the source code, you can do it with copy/paste commands. Please do not use screenshots for code listings.**
4. **Other screenshots (figures, graphs, etc.) should be scaled appropriately. Please cut off unnecessary elements on the images.**

© BME – AAIT 2015