

# 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	<input checked="" type="checkbox"/>
TASK 2	<input checked="" type="checkbox"/>
TASK 3	<input checked="" type="checkbox"/>
TASK 4	<input checked="" type="checkbox"/>
TASK 5	<input checked="" type="checkbox"/>
TASK 6	<input checked="" type="checkbox"/>
TASK 7	<input checked="" type="checkbox"/>
TASK 8	<input checked="" type="checkbox"/>
TASK 9	<input checked="" type="checkbox"/>
TASK 10	<input checked="" type="checkbox"/>
TASK 11	<input checked="" type="checkbox"/>
TASK 12	<input checked="" type="checkbox"/>
TASK 13	<input checked="" type="checkbox"/>

### 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:**

Query01\_Transfer...9T2UOC\Meng (52))

```
1 CREATE DATABASE lab4;
2
3 USE lab4;
4
5 SELECT *
6 FROM account;
7
8 -- TASK 01
9
10 create table account
11 (id int primary key,
12 balance int)
13 insert into account values(1
14 insert into account values(2
15 insert into account values(4
16
17 -- TASK 02
18 -- 2 UPDATES HAPPENED INDEPE
19
20 update account
21 set balance=balance-1000
```

81 %

Results Messages

	id	balance
1	1	2500
2	2	11000
3	4	6400

**Reasoning:**

I just created a table as the guide said.

### 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:

Query01\_Transfer...9T2UOC\Meng Query01\_Transfer...9T2UOC\Meng Query01\_Transfer...9T2UOC\Meng (76))

1  
2  
3 -- TASK 02  
4  
5 select \* from  
6 account;

39 % 89 % 89 %

Results Messages Results Messages Results Messages

id	balance
1	5000
2	8500
3	6400

Before transaction

id	balance
1	4000
2	8500
3	6400

At the middle of transaction

id	balance
1	4000
2	9500
3	6400

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.

### 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:**

The image displays three screenshots of SQL Server Enterprise Manager, showing the execution of a script that demonstrates a deadlock scenario. The script involves two tasks: Task 03 and Task 04. Task 03 updates the balance of account 1, and Task 04 updates the balance of account 2. The script is designed to show a deadlock when the first transaction is not committed before the second transaction starts.

**Before an update**

Query01\_Transfer...9T2UOC\Meng

```
28 where id=2;
29
30
31 -- TASK 03
32 -- IN THIS WAY THE TRANSACTIONS
33 -- IF WE DON'T COMMIT,
34
35 begin transaction
36 update account
37 set balance=balance-1000
38 where id=1;
39
40 update account
41 set balance=balance+1000
42 where id=2;
43 commit;
44
45 -- TASK 04
46 -- there is a deadlock
47 -- transfer 1 successful
```

Results

id	balance
1	4000
2	9500
3	6400

**Between the update**

Query01\_Transfer...9T2UOC\Meng

```
15
16 UPDATE account
17 SET balance = balance - 1000
18 WHERE id = 1;
19
20 COMMIT;
```

Results

id	balance
1	3000
2	10500
3	6400

**After the commit**

Query01\_Transfer...9T2UOC\Meng

```
1
2 -- TASK 02
3
4 select * from
5 account;
6
7
8 -- TASK 04
9
10 BEGIN TRANSACTION
11
12 UPDATE account
13 SET balance = balance - 1000
14 WHERE id = 2;
15
16 UPDATE account
17 SET balance = balance + 1000
18 WHERE id = 1;
19
20 COMMIT;
```

Results

id	balance
1	3000
2	10500
3	6400

**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.

### 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:**

The visual result shows three stages of the database state:

- Before transaction:** The database state is shown with two accounts. Account 1 has a balance of 3000, and Account 2 has a balance of 10500.
- Transaction 2 is aborted due to deadlock:** The screenshot shows the error message: "Msg 1205, Level 13, State 51, Line 13 Transaction (Process ID 53) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Completion time: 2022-04-06T02:00:04.9409838-07:00".
- After transaction:** The database state is shown after the deadlock is resolved. Account 1 has a balance of 2500, and Account 2 has a balance of 11000.

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.

### 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:**

The screenshot shows a SQL script being executed in a query window. The script creates two tables, 'exam' and 'signup', and inserts data into them. Below the script, the 'Results' pane shows the data inserted into the 'exam' table. The 'Messages' pane shows the execution progress.

examid	studentid
1	111
2	222

  

id	subject	date	limit
1	Informatics2	2007-06-15 00:00:00.000	3
2	Mathematics	2007-06-18 00:00:00.000	3

**Reasoning:**

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

## 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	<code>select limit from exam where id=1;</code>	
2		<code>select limit from exam where id=1;</code>
3	<code>select count(*) from signup where examid=1;</code>	
4		<code>select count(*) from signup where examid=1;</code>
5	<code>insert into signup values(1, 333);</code>	
6		<code>insert into signup values(1, 444);</code>

**Visual result:**

```

Query01_Transfer...9T2UOC\Meng (52)*
84 values(2, 'Mathematics', convert(
85
86 insert into signup values(1,111)
87 insert into signup values(1,222)
88
89 SELECT *
90 FROM signup;
91
92 SELECT *
93 FROM exam;
94
95 -- TASK 06
96 -- TWO STUDENT SEE THAT FOR THIS
97 -- THIS IS PHANTOM RECORD OR (RE
98
99 select limit
100 from exam
101 where id=1;
102
89 %
Results Messages
examid studentid
1 1 111
2 1 222

id subject date limit
1 1 Informatics2 2007-06-15 00:00:00.000 3
2 2 Mathematics 2007-06-18 00:00:00.000 3

```

State before transactions

```

Query01_Transfer...9T2UOC\Meng (52)*
93 FROM exam;
94
95 -- TASK 06
96 -- TWO STUDENT SEE THAT FOR THIS
97 -- THIS IS PHANTOM RECORD OR (RE
98
99 select limit
100 from exam
101 where id=1;
102
103 select count(*)
104 from signup
105 where examid=1;
106
107 insert into signup
108 values(1, 333);
109
110 -- TASK 07
111
89 %
Results Messages
examid studentid
1 1 111
2 1 222
3 1 333
4 1 444

id subject date limit
1 1 Informatics2 2007-06-15 00:00:00.000 3
2 2 Mathematics 2007-06-18 00:00:00.000 3

```

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.

## 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:**

The image shows three screenshots of SQL Server Enterprise Manager, illustrating the execution of transactions and the resulting data in the 'signup' table.

**Query01\_Transfer...9T2UOC\Meng (51):** This query shows the initial state of the 'signup' table with 4 rows: (1, 111), (1, 222), (1, 333), and (1, 444).

**Query01\_Transfer...9T2UOC\Meng (52):** This query shows the execution of a transaction. The first part of the transaction (lines 40-50) is highlighted in yellow. The second part (lines 51-56) is highlighted in blue. The transaction is committed.

**Query02\_Transfer...9T2UOC\Meng (53):** This query shows the execution of a transaction. The first part of the transaction (lines 121-125) is highlighted in yellow. The second part (lines 126-130) is highlighted in blue. The transaction is committed.

**Results:** The results of the queries are shown in the 'Results' pane. The 'signup' table now contains 5 rows: (1, 111), (1, 222), (1, 333), (1, 444), and (1, 555).

**Messages:** The messages pane shows the following messages:

- Msg 1205, Level 13, State 46, Line 51: Transaction (Process ID 53) was deadlocked on lock resources with another process and has been chosen as the deadlock victim.

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.



## 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	<code>UPDATE exam SET limit = 5 WHERE exam.id = 1;</code>	
2	<code>set transaction isolation level serializable; Begin transaction</code>	
3		<code>set transaction isolation level serializable; Begin transaction</code>
4	<code>select limit from exam where id=1;</code>	
5		<code>select limit from exam where id=1;</code>
6	<code>select count(*) from signup where examid=1;</code>	
7		<code>select count(*) from signup where examid=1;</code>
8	<code>insert into signup values(1,555);</code>	
9		<code>insert into signup values(1,666);</code>

**Visual result:**

The visual result consists of three screenshots of SQL query results:

- Left Screenshot:** Shows the initial state of the database. The 'exam' table has one row with id=1 and limit=5. The 'signup' table has three rows with examid=1 and studentid=111, 333, and 444.
- Middle Screenshot:** Shows the state after the first transaction (Task 08) is executed. The 'exam' table now has limit=5. The 'signup' table has four rows, including the new row with examid=1 and studentid=555.
- Right Screenshot:** Shows the state after the second transaction (Task 09) is executed. The 'exam' table still has limit=5. The 'signup' table has five rows, including the new row with examid=1 and studentid=666.

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.

## 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:**

The screenshot displays two SQL Server query windows. The left window, 'Query01\_Transfer...9T2UOC\Meng (52)', shows a successful transaction for student 111. The right window, 'Query01\_Transfer...9T2UOC\Meng (57)', shows a deadlock error for student 67. The error message states: 'Msg 1205, Level 13, State 48, Line 101: Transaction (Process ID 67) was deadlocked on lock resources with another process'. The completion time is 2022-04-08T08:36:44.8100650-07:00.

**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.

### 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:**

Query01\_Transfer...9T2UOC\Meng (52)\*

```
82
83 insert into exam
84 values(2, 'Mathematics',conve
85
86 insert into signup values(1,1
87 insert into signup values(1,1
88
89 SELECT *
90 FROM signup;
91
92 SELECT *
93 FROM exam;
94
95 -- TASK 06
96
```

99 %

Results Messages

	examid	studentid
1	1	111
2	1	222

  

	id	subject	date	limit
1	1	Informatics2	2007-06-15 00:00:00.000	3
2	2	Mathematics	2007-06-18 00:00:00.000	3

**Reasoning:**

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

### 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:**

Query01\_Transfer...9T2UOC\Meng (52))

```

220
221 SELECT *
222 FROM exam;
223
224 -- TASK 11
225
226 set transaction isolation
227 level read committed
228 begin transaction
229
230 select limit
231 from exam with(XLOCK)
232 where id=1
233
234 select count(*)
235 from signup
236 where examid=1
237
238 insert into signup

```

89 %

examid	studentid
1	111
2	222

id	subject	date	limit
1	Informatics2	2007-06-15 00:00:00.000	3
2	Mathematics	2007-06-18 00:00:00.000	3

State before transaction

Query01\_Transfer...9T2UOC\Meng (52))

```

214
215 insert into signup values(1,1
216 insert into signup values(1,2
217
218 SELECT *
219 FROM signup;
220
221 SELECT *
222 FROM exam;
223
224 -- TASK 11
225
226 set transaction isolation
227 level read committed
228 begin transaction
229
230 select limit
231 from exam with(XLOCK)
232 where id=1

```

89 %

examid	studentid
1	111
2	222
3	333

id	subject	date	limit
1	Informatics2	2007-06-15 00:00:00.000	3
2	Mathematics	2007-06-18 00:00:00.000	3

State after transaction

### 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.

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

**Visual result:**

Query01\_Transfer...9T2UOC\Meng (52)\*

```

241 commit
242
243 -- TASK 12
244
245 UPDATE exam
246 SET limit = 6
247 WHERE id = 1;
248
249 SELECT *
250 FROM signup;
251
252 SELECT *
253 FROM exam;
254
255 set transaction isolation
256 level read committed
257 begin transaction
258
259 select limit
  
```

Results

examid	studentid
1	111
2	222

  

id	subject	date	limit
1	Informatics2	2007-06-15 00:00:00.000	6
2	Mathematics	2007-06-18 00:00:00.000	3

State before transactions

Query01\_Transfer...9T2UOC\Meng (52)\*

```

256 level read committed
257 begin transaction
258
259 select limit
260 from exam with(XLOCK)
261 where id=1
262
263 select count(*)
264 from signup
265 where examid=1
266
267 insert into signup
268 values(1,333)
269
270 commit
  
```

Results

examid	studentid
1	111
2	222
3	333
4	444

  

id	subject	date	limit
1	Informatics2	2007-06-15 00:00:00.000	6
2	Mathematics	2007-06-18 00:00:00.000	3

State after transaction

### 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.

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

**Visual result:**

Query01\_Transfer...9T2UOC\Meng (52)\*

```

275 --level read committed
276 begin transaction
277
278
279 --select limit
280 from exam with(XLOCK)
281 where id=1
282
283 --select count(*)
284 from signup
285 where examid=1
286
287 --insert into signup
288 values(1,333)
289
290 commit
291

```

examid	studentid
1	111
2	222

id	subject	date	limit
1	Informatics2	2007-06-15 00:00:00.000	3
2	Mathematics	2007-06-18 00:00:00.000	3

State before transactions

Query01\_Transfer...9T2UOC\Meng (52)\*

```

278
279 --select limit
280 from exam with(XLOCK)
281 where id=1
282
283 --select count(*)
284 from signup
285 where examid=1
286
287 --insert into signup
288 values(1,333)
289
290 commit
291
292 --SELECT *
293 FROM signup
294
295 --SELECT *
296 FROM exam
297

```

examid	studentid
1	111
2	222
3	333
4	444

id	subject	date	limit
1	Informatics2	2007-06-15 00:00:00.000	3
2	Mathematics	2007-06-18 00:00:00.000	3

State after transaction



### 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.

### 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.**