LABORATORY REPORT

LABORATORY REPORT

COMMON DATA	
STUDENT NAME	**DAS ANJAN KUMAR**
NEPTUN CODE	**D42DQA**
DEPARTMENT	**DEPT. OF AUTOMATION AND APPLIED INFORMATICS**
INSTRUCTOR NAME	**AL MAGSOOSI HUSAM**
LABORATORY PLACE	**IL206**
LABORATORY TIME	**6TH APRIL 2022**
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	☐

EXERCISES

TASK #1

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

Solution:

SQLQuery1.sql - (L...L5KV2R\Anjan (63))*

```
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)
```

110 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2022-04-12T19:52:56.6532810+02:00
```

DESKTOP-NL5KV2R\...b 4 - dbo.account

	id	balance
	1	5000
	2	8500
	4	6400
*	NULL	NULL

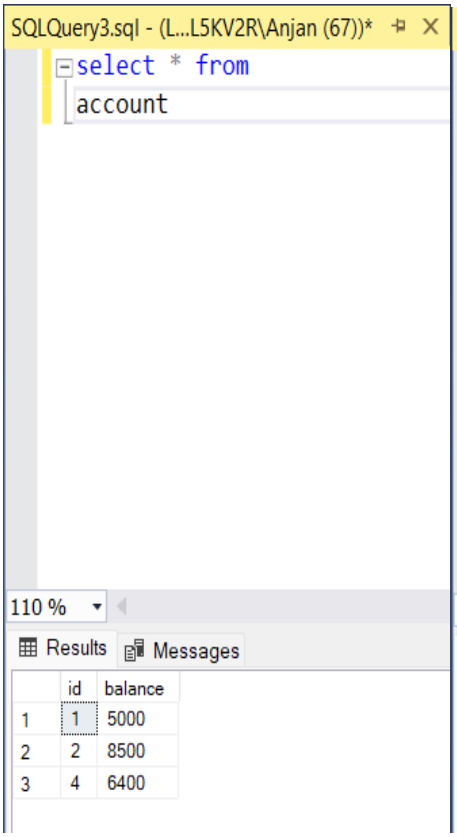
Reasoning: After creating a new database for Lab 4, I created a new query and ran the above mentioned codes. As a result we god the simple table of 2 columns and 3 rows.

TASK #2

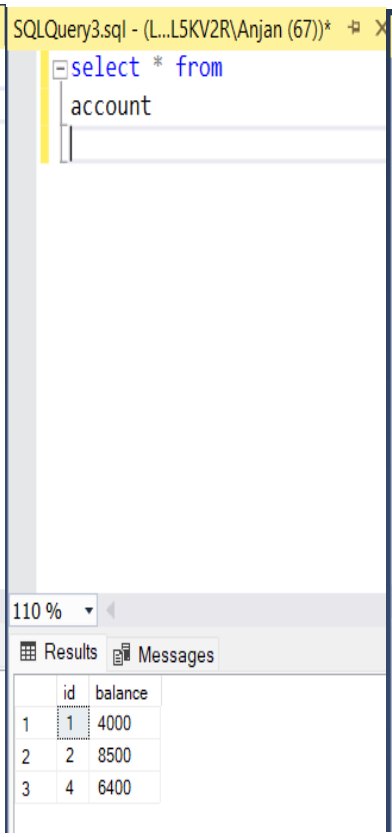
Problem statement: Transfer 1000 HUF from bank account 1 to 2. In order to check the transfer open another query window, this creates a second parallel connection to the database. Check balance during transfer continuously. What did you notice and why?

Solution:

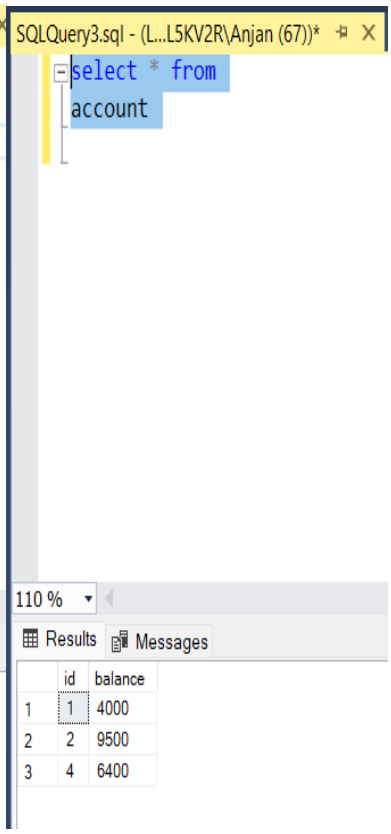
1



2



3



Reasoning:

This tasks demonstrates the **atomicity** property of database transaction and these properties are very important to ensure the accuracy of its completeness and data integrity. We know that States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction. But here we check the data in between a transaction (step 2) which creates a misinformation, because at that time the 1000 HUF is no where, we subtract it from one account, but did not send it to the other account, which causes faulty data or misinterpretation. So, we should not access the database in between a transaction, this checking always should be before or after the transaction.

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:

SQLQuery3.sql - (Loc...n (67)) Executing...*

```
begin transaction

update account
set balance=balance-1000
where id=1

update account
set balance=balance+1000
where id=2
```

Messages

```
(1 row affected)

(1 row affected)

Completion time: 2022-04-12T20:14:56.1090009+02:00
```

SQLQuery3.sql - (Loc...n (67)) Executing...*

```
select * from
account
```

Results | Messages

Executing query...

SQLQuery3.sql - (L...L5KV2R\Anjan (67))*

```
/*begin transaction

update account
set balance=balance-1000
where id=1

update account
set balance=balance+1000
where id=2 */

commit transaction
```

Messages

```
Commands completed successfully.

Completion time: 2022-04-12T20:16:25.3912503+02:00
```

SQLQuery3.sql - (L...L5KV2R\Anjan (67))*

```
select * from
account
```

Results | Messages

	id	balance
1	1	4000
2	2	9500
3	4	6400

Query executed successfully.

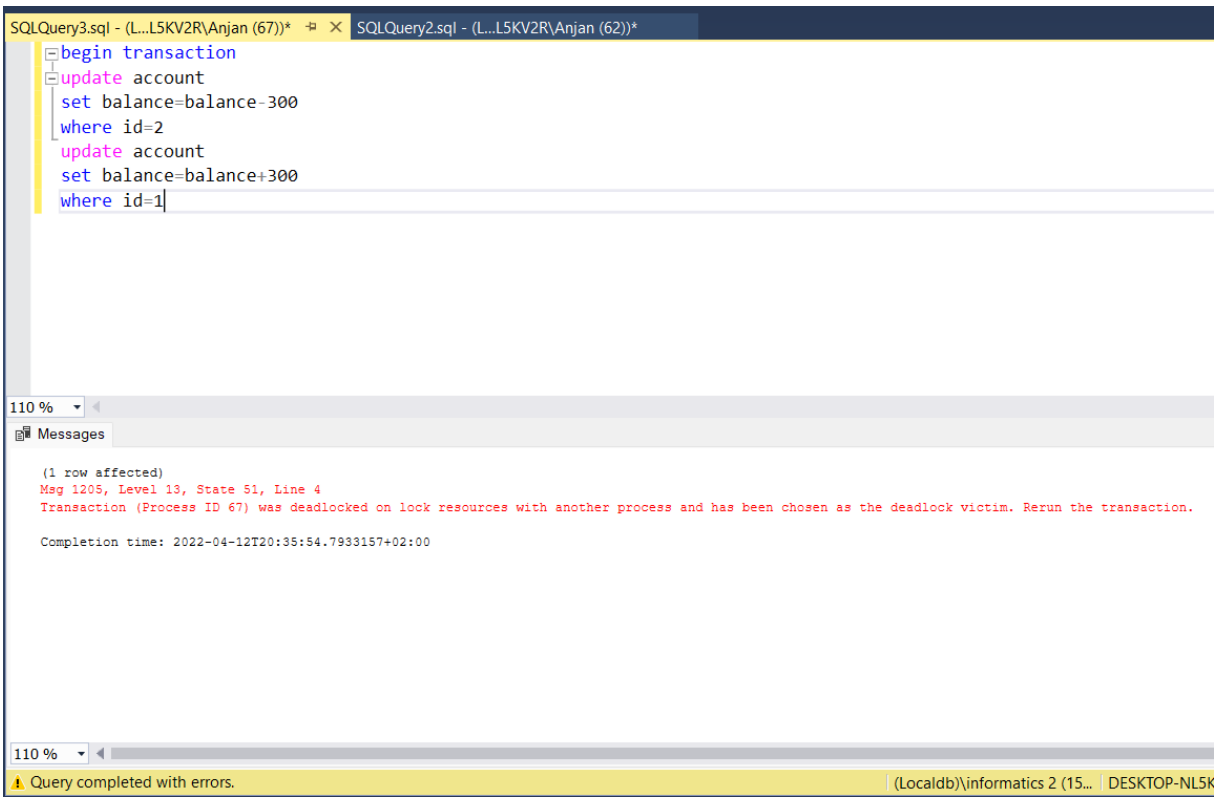
Reasoning:

Here I have used the begin transaction – commit transaction keyword pair in order to deny any other query to access the database while the transaction Is taking place. We can see here in the upper two images, that I did not write the commit transaction first, as a result, when I tried to check the databases it was showing "Executing Query" and this only turned into "Query executed successfully" only when I used commit transaction. So, we always should use the keyword pair in order to ensure the atomic property of transaction.

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:



Reasoning:

This task is a perfect demonstration of deadlock, as we can see from the error it is written that "`Transaction (Process ID 67) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.`" It is because noth of the transactions are waiting for each other; that means that none of them can continue running. W Each of them is waiting for another transaction means waiting for a resource used by the other one.

TASK #5

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

Solution:

SQLQuery4.sql - (L...L5KV2R\Anjan (56))*

```sql
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)
```

110 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2022-04-12T20:45:49.3731311+02:00
```

DESKTOP-NL5KV2R\...ab 4 - dbo.signup

	examid	studentid
▶	1	111
	1	222
*	NULL	NULL

DESKTOP-NL5KV2R\...ab 4 - dbo.signup | DESKTOP-NL5K

	id	subject	date	limit
▶	1	Informatics2	2007-06-15 ...	3
	2	Mathematics	2007-06-18 ...	3
*	NULL	NULL	NULL	NULL

Reasoning:

Two tables named exam and sign up are created with proper code execution.

TASK #6

Problem statement:

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

Solution:

SQLQuery9.sql - (L...L5KV2R\Anjan (65))* | SQLQuery8.sql - (L...L5KV2R\Anjan (61))*

```
--select limit from exam where id=1
--select count(*) from signup where examid=1

insert into signup values(1, 333)
```

110 %

Messages

```
(1 row affected)

Completion time: 2022-04-12T21:49:06.4131330+02:00
```

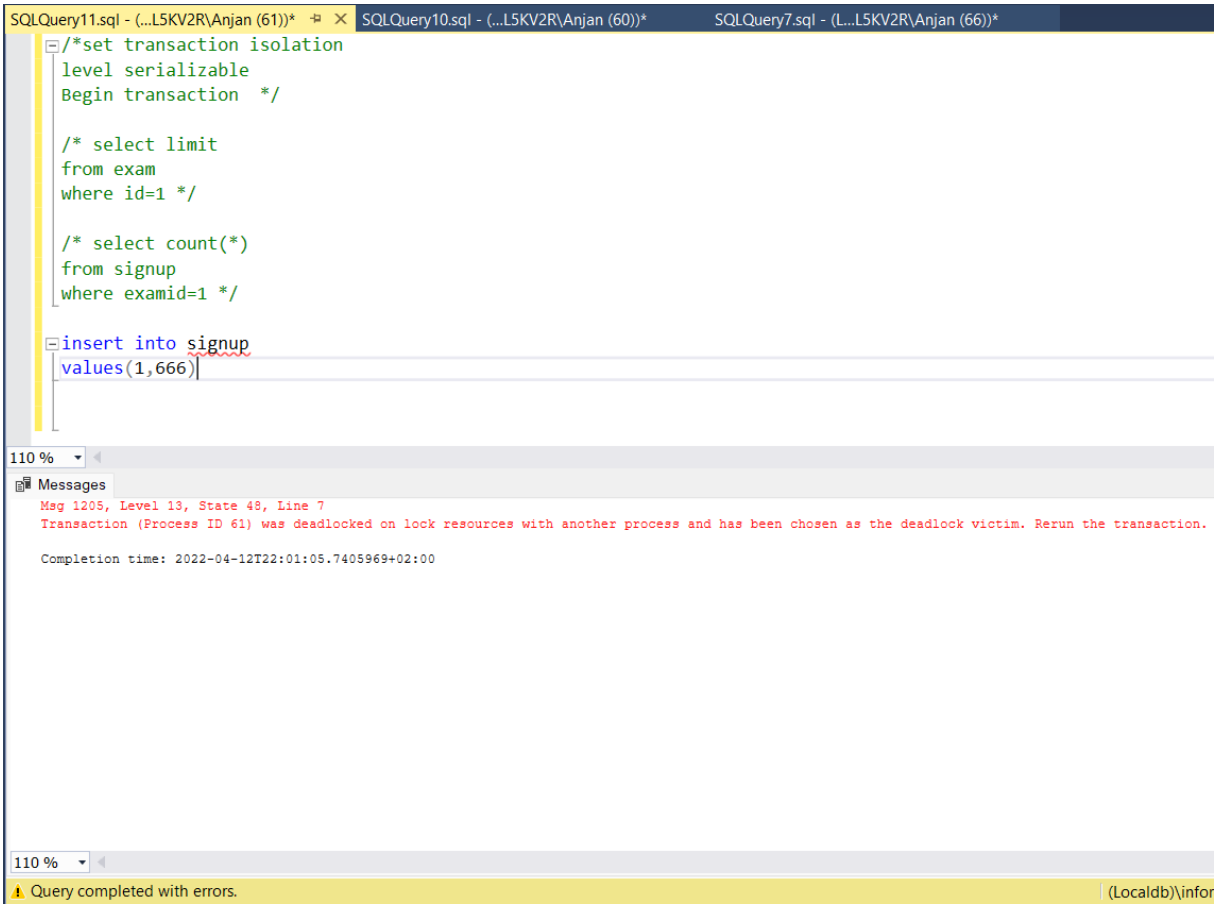
Reasoning:

Here, we don't find any errors defined by the system, but there can be a problem happening is that as two students are trying to sign up for the same exam at the same time , as a result irrelevant data are taking place, because there are only one empty space for registration, but both of them register. This can be regarded as phantom error.

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. What is the result? Why?

Solution:



Reasoning:

Even by using isolation we could not solve this problem as deadlock occurs. This is because these two transactions are dependent on each other. It is because both transactions are being executed and updating data to the same row at the same time. Also there is the fact there were only two seats remaining , as a result when both of the transaction want to execute the error happens.

TASK #8

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

Solution:

DESKTOP-NL5KV2R\....Lab 4 - dbo.exam | SQLQuery11.sql -

	id	subject	date	limit
	1	Informatics2	2007-06-15 ...	5
	2	Mathematics	2007-06-18 ...	3
*	NULL	NULL	NULL	NULL

SQLQuery3.sql - (L...L5KV2R\Anjan (56))* | SQLQuery2.sql - (L...L5KV2R\Anjan (55))*

```
/* set transaction isolation
level serializable
Begin transaction

select limit
from exam
where id=1

select count(*)
from signup
where examid=1 */

insert into signup
values(1,555)
```

110 %

Messages

```
Msg 2627, Level 14, State 1, Line 11
Violation of PRIMARY KEY constraint 'PK__signup__F1BD43412AB5DD03'. Cannot insert duplicate key in object 'dbo.signup'. The duplicate key value is (1, 555).
The statement has been terminated.

Completion time: 2022-04-12T22:17:14.9038808+02:00
```

SQLQuery3.sql - (L...L5KV2R\Anjan (56))* | SQLQuery2.sql - (L...L5KV2R\Anjan (55))*

```
/*set transaction isolation
level serializable
Begin transaction

select limit
from exam
where id=1

select count(*)
from signup
where examid=1 */

insert into signup
values(1,666)
```

110 %

Messages

```
Msg 1205, Level 13, State 48, Line 11
Transaction (Process ID 55) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

Completion time: 2022-04-12T22:17:14.8548554+02:00
```

Reasoning:

I increased the limit to 5 first and then executed the codes again, but now we can see two different errors in two queries. In the first one we can see a duplicate key error. The primary key is already protecting us from inserting duplicate values, as we are experiencing when we get that error. Adding another unique constraint isn't necessary to do that. The "duplicate key" error is telling us that the work was not done because it would produce a duplicate key, not that it discovered a duplicate key already commited to the table.

And on the other hand we are experiencing deadlock error because the second task is waiting for the first query.

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:

SQLQuery3.sql - (L...L5KV2R\Anjan (62))* | SQLQuery2.sql - (L...L5KV2R\Anjan (56))*

```
--set transaction isolation
--level serializable
--Begin transaction

--select limit
--from exam
--where id=1

--select count(*)
--from signup
--where examid=1

insert into signup
values(1,555)
```

110 %

Messages

```
Msg 1205, Level 13, State 48, Line 13
Transaction (Process ID 62) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

Completion time: 2022-04-12T22:35:49.1500153+02:00
```

SQLQuery3.sql - (L...L5KV2R\Anjan (62))* | SQLQuery2.sql - (L...L5KV2R\Anjan (56))*

```
--set transaction isolation
--level serializable
--Begin transaction

--select limit
--from exam
--where id=2

--select count(*)
--from signup
--where examid=2

insert into signup
values(2,666)
```

110 %

Messages

```
(1 row affected)

Completion time: 2022-04-12T22:35:49.0967109+02:00
```

110 %

Query executed successfully.

Reasoning:

In this case , when we changed the sign up of one student to another exam, we could avoid the duplicate key error problem, as we are not trying to write the same data into same field at the same time. But we could not avoid the deadlock problem, because one problem was waiting for the execution of the other one. The one sign up was executed successfully because the deadlock of the other signup didn't affect the first signup because of the isolation of the transactions.

TASK #10

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

Solution:

SQLQuery2.sql - (L...L5KV2R\Anjan (56))*

```sql
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)
```

110 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2022-04-12T22:41:40.7843312+02:00
```

Reasoning:

Executed the code again and reset the database to the original .

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:

SQLQuery4.sql - (L...L5KV2R\Anjan (59))*

```
--set transaction isolation
--level read committed
--begin transaction

--select limit
--from exam with(XLOCK)
--where id=1

--select count(*)
--from signup
--where examid=1

commit
```

110 %

Messages

```
Commands completed successfully.

Completion time: 2022-04-12T22:47:51.5112370+02:00
```

SQLQuery5.sql - (L...L5KV2R\Anjan (60))*

```
--set transaction isolation
--level read committed
--begin transaction

--select limit
--from exam with(XLOCK)
--where id=1

--select count(*)
--from signup
--where examid=1

--insert into signup
--values(1,333)

commit
```

110 %

Messages

```
Commands completed successfully.

Completion time: 2022-04-12T22:47:07.9908605+02:00
```

Reasoning:

In this case we could see that both of the transaction resulted in successful execution. This is because of the use of level read committed and XLOCK. Because in this way one student keeps waiting until the other students does not finish the transaction (commit). As a result there is no possibility of any double key error. Because the other student will only be able to commit transaction after the first student is done.

TASK #12

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

Solution:

DESKTOP-NL5KV2R\....Lab 4 - dbo.exam | SQLQuery5.sql - (L

	id	subject	date	limit
	1	Informatics2	2007-06-15 ...	6
	2	Mathematics	2007-06-18 ...	3
▶*	NULL	NULL	NULL	NULL

SQLQuery5.sql - (L...L5KV2R\Anjan (60))* | SQLQuery4.sql - (L...L5KV2R\Anjan (59))*

```
--set transaction isolation
--level read committed
--begin transaction

--select limit
--from exam with(XLOCK)
--where id=1

--select count(*)
--from signup
--where examid=1

insert into signup
values(1,333)

--commit
```

110 %

Messages

```
Msg 2627, Level 14, State 1, Line 13
Violation of PRIMARY KEY constraint 'PK__signup__F1BD43413C0EB39E'. Cannot insert duplicate key in object 'dbo.signup'. The duplicate key value is (1, 333).
The statement has been terminated.

Completion time: 2022-04-12T23:02:40.1156327+02:00
```

SQLQuery5.sql - (L...L5KV2R\Anjan (60))* | SQLQuery4.sql - (L...L5KV2R\Anjan

```
--set transaction isolation
--level read committed
--begin transaction

--select limit
--from exam with(XLOCK)
--where id=1

--select count(*)
--from signup
--where examid=1

--insert into signup
--values(1,333)

commit
```

110 %

Messages

```
Commands completed successfully.

Completion time: 2022-04-12T23:05:33.5075938+02:00
```

Reasoning:

Firstly, if we don't run the table creating commands again or don't remove the previous value that was created in the previous task, we will see the duplicate key error, because (1,333) was created previously. Apart from that, if we don't consider this error, then the message of this task is that when another student will try to access the same exam, but he will be in waiting because of the other transaction happening with level read commit and xlock. But when the other transaction is finished

this student will realize that he also can register for the same exam as limit of the exam has been increased.

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:

SQLQuery2.sql - (L...L5KV2R\Anjan (66))* | SQLQuery3.sql - (L...L5KV2R\Anjan (68))*

```
--set transaction isolation
--level read committed
--begin transaction

select limit
from exam with(XLOCK)
where id=2

--select count(*)
--from signup
--where examid=2

--commit
```

110 %

Results | Messages

	limit
1	3

Reasoning:

In this task, we can see the second sign up does not wat to execute for the first sign up. It is because they are signing up two different exam. So, we should realize this feature that it only make wait an execution, only when the are dedicated to the same dataset. When the queries are not interfering with each other then there will be no problem at all.

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.**