EXERCISES

TASK #1

Problem statement:

We need to connect to the SQL Server and create a bank account database using the following SQL script.

Solution:

Copying the following code :

Code:

```
Use lab4
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)
```

Reasoning: Result:

We have got an account table with different balances for each account.

TASK #2

Problem statement:

We need to create a transaction to transfer 1000 HUF from bank account n°1 to n°2.
In order to check the transfer open another query window, this creates a second parallel connection to the database.

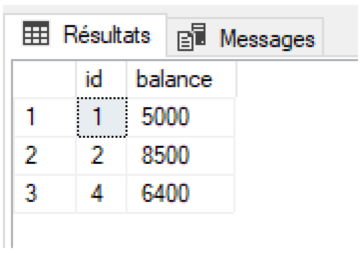
Solution:

By Data Definition Language scripts :
We use UPDATE command to modify the existing records in the account table.
And in order to specify which columns and values that should be updated in a table we use SET command :
* We substract from the balance of the 1st account by 1000huf.
* Then we add 1000huf to the balance of the 2nd account.

Code:

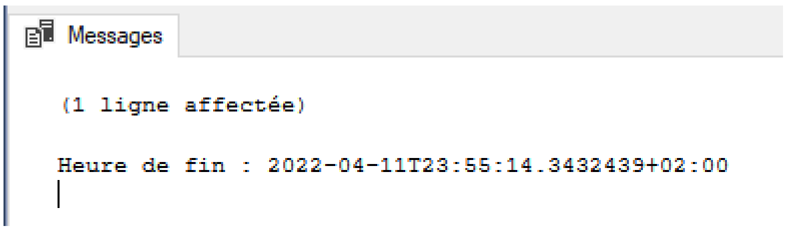
```
(1)      select * from account

(2)      use lab4
         update account
         set balance=balance-1000
         where id=1

(3)      select * from account

(4)      use lab4
         update account
         set balance=balance+1000
         where id=2

(5)       select * from account
```

Reasoning: Result:



Résultats | Messages

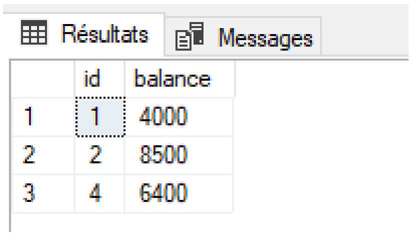
	id	balance
1	1	5000
2	2	8500
3	4	6400



Messages

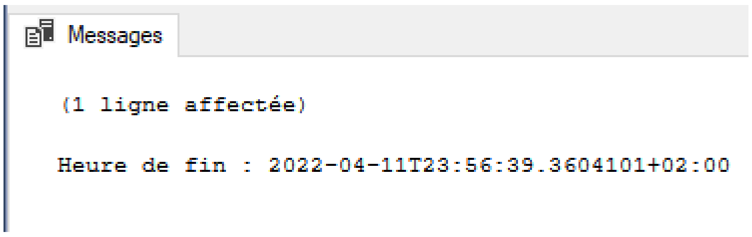
```
(1 ligne affectée)

Heure de fin : 2022-04-11T23:55:14.3432439+02:00
```

(1) **(2)**



Résultats | Messages

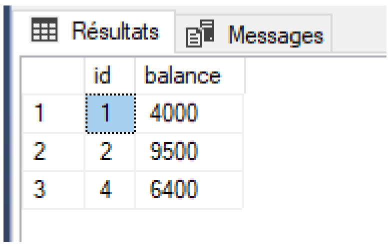
	id	balance
1	1	4000
2	2	8500
3	4	6400



Messages

```
(1 ligne affectée)

Heure de fin : 2022-04-11T23:56:39.3604101+02:00
```

(3) **(4)**



Résultats | Messages

	id	balance
1	1	4000
2	2	9500
3	4	6400

(5)

We have got an updated table with a new balance for each operation.
We notice that a 1000huf was transferred from the 1st row (account 1) to the 2nd row (account 2)

TASK #3

Problem statement:

We need to do again the second task by changing the script in a way that the data manipulation operations are put into the same transaction.

Solution:

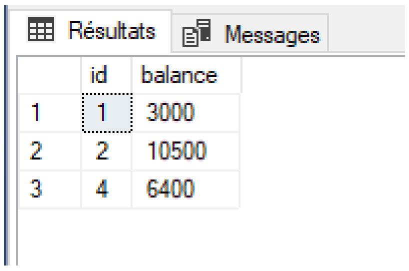
By Data Definition Language scripts :

Code:

```
Use lab4
begin transaction
update account
set balance=balance-1000 where id=1
update account
set balance=balance+1000 where id=2
```

Reasoning:

Result:



	id	balance
1	1	3000
2	2	10500
3	4	6400

We notice that we got a different result compared to the previous task, that is because the transaction has not yet been committed while the system were trying to read the data. That is called Dirty read.

TASK #4

Problem statement:

We need to do two transfers 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.

Solution:

By Data Definition Language scripts :
We copy the following code

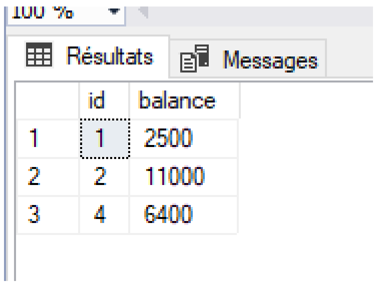
Code:

First transaction	Second transaction

`use lab4`
`begin transaction`
`update account`
`set balance=balance-500`
`where id=1`
`update account`
`set balance=balance+500`
`where id=2`
	`use lab4`
	`begin transaction`
	`update account`
	`set balance=balance-300`
	`where id=1`
	`update account`
	`set balance=balance+300`
	`where id=2`

First transaction **Second transaction**

Reasoning: Result:



```
   id  balance
1  1   2500
2  2   11000
3  4   6400
```

Each transaction cannot continue and begins to wait for the other one, that is why we got the same result. That is called deadlock.

TASK #5

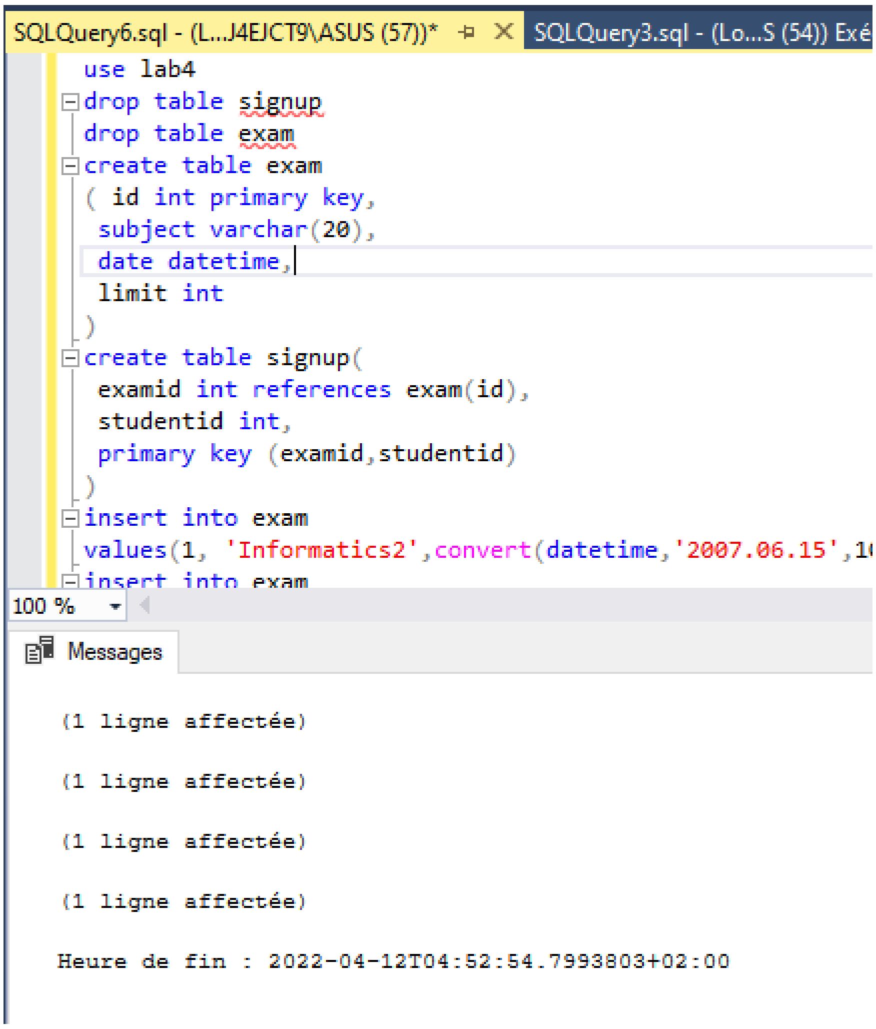
Problem statement:

We need to create a simple exam signup system using the following SQL script:

Solution:

By Data Definition Language scripts :
We need to copy the following code

Code and Result:


SQLQuery6.sql - (L...J4EJCT9\ASUS (57))* SQLQuery3.sql - (Lo...S (54)) Exé

```
use lab4
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',1
insert into exam
```

100 %

Messages

```
(1 ligne affectée)

(1 ligne affectée)

(1 ligne affectée)

(1 ligne affectée)

Heure de fin : 2022-04-12T04:52:54.7993803+02:00
```

Reasoning:

We have got table of an exam signup system.

TASK #6

Problem statement:

We need to simulate two concurrent signups to the first exam.

Solution:

By Data Definition Language scripts :

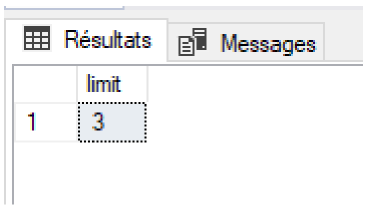
We need to copy the following code

Code:

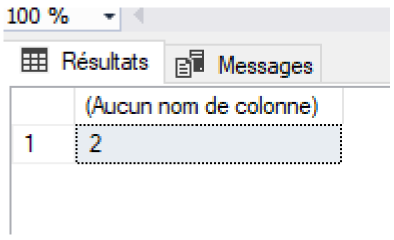
(1) `use lab4` `select limit` `from exam` `where id=1` (2) `use lab4` `select count(*)` `from signup` `where examid=1` (3) `use lab4` `insert into signup` `values(1, 333)`	(1) `use lab4` `select limit` `from exam` `where id=1` (2) `use lab4` `select count(*)` `from signup` `where examid=1` (3) `use lab4` `insert into signup` `values(1, 444)`
First signup	Second signup

Reasoning:

Result:


Résultats | Messages

```
  limit
1 3
```


100 %
Résultats | Messages

```
  (Aucun nom de colonne)
1 2
```

We have got an error ,since we were repeating a read operation on the same records, but we got different records in the results set. This is called phantom read

TASK #7

Problem statement:

We need to do again the previous task by a different method

Solution:

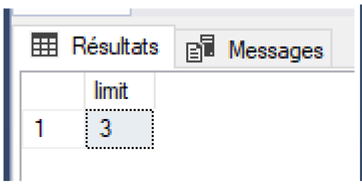
We need to copy the new code and increase isolation levels of transactions to "serializable"

Code:

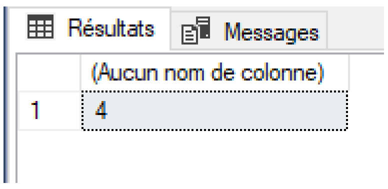
(1) set transaction isolation level serializable Begin transaction (2) select limit from exam where id=1 (3) select count(*) from signup where examid=1 (4) insert into signup values(1,555)	(1) set transaction isolation level serializable Begin transaction (2) select limit from exam where id=1 (3) select count(*) from signup where examid=1 (4) insert into signup values(1,666)
1st signup	2nd signup

Reasoning:

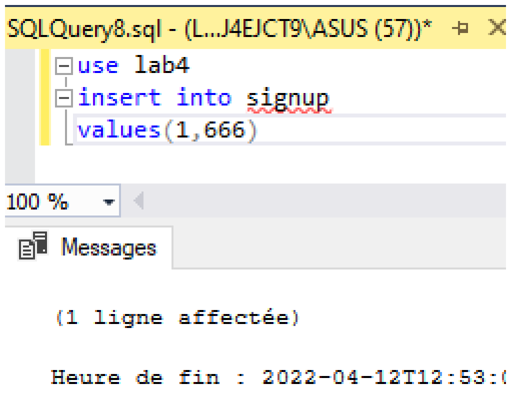
Result:



```
   limit
1  3
```



```
   (Aucun nom de colonne)
1  4
```



```
use lab4
insert into signup
values(1,666)
```

```
(1 ligne affectée)

Heure de fin : 2022-04-12T12:53:(
```

The written values were added successfully

As we used serializable level, both of the transactions behave as if they were executed one after the other

TASK #8

Problem statement:

We need to increase limit of the first exam to 5, then do again the task7.

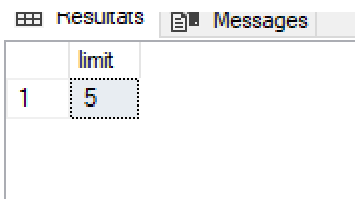
Solution:

Code: increase limit of the first exam to 5

```
use lab4
update exam set limit=5 where id=1
```

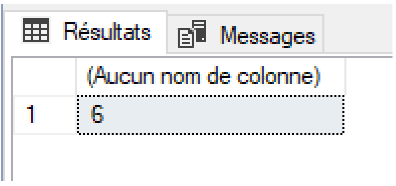
Reasoning:

Result:



Résultats | Messages

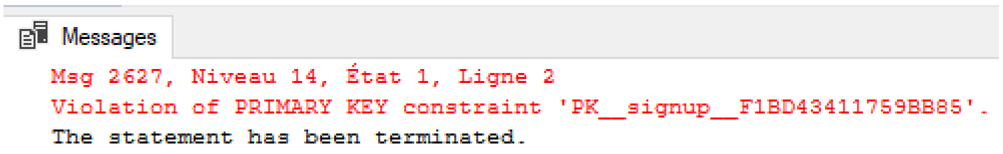
	limit
1	5



Résultats | Messages

	(Aucun nom de colonne)
1	6

We have got an increased limit and count.

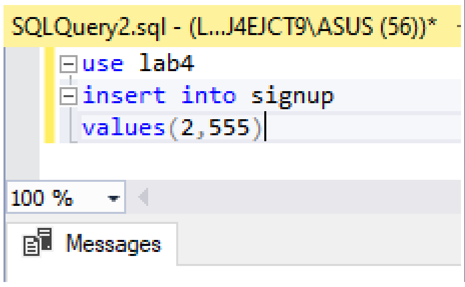


Messages

```
Msg 2627, Niveau 14, État 1, Ligne 2
Violation of PRIMARY KEY constraint 'PK__signup__F1BD43411759BB85'.
The statement has been terminated.
```

We have got an error because we can not insert the same value multiple times.

That is why we have to change the inserted value to (2,555)and(2,666) because we already used (1,555) and(1,666) because we can not insert the same value multiple times.

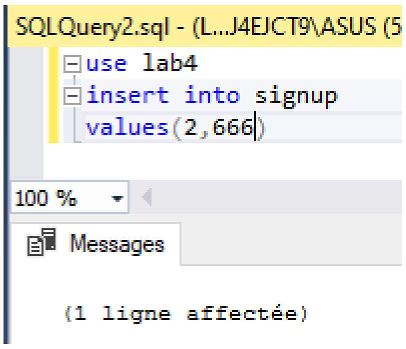


SQLQuery2.sql - (L...J4EJCT9\ASUS (56))*

```
use lab4
insert into signup
values(2,555)
```

100 %

Messages



SQLQuery2.sql - (L...J4EJCT9\ASUS (5

```
use lab4
insert into signup
values(2,666)
```

100 %

Messages

```
(1 ligne affectée)
```

TASK #9

Problem statement:

We need to do again the task 7 with a different method :the first student should signup to the first exam while the second one should signup to the second exam.

Solution:

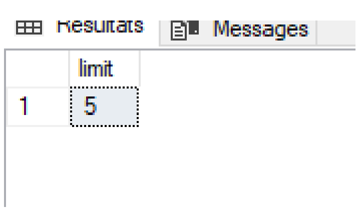
We need to recopy the code from the task 7

Code:

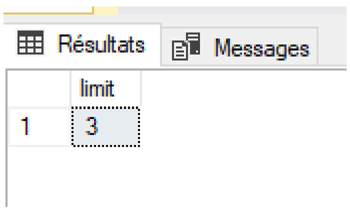
```
use lab4
```

Reasoning:

Result:

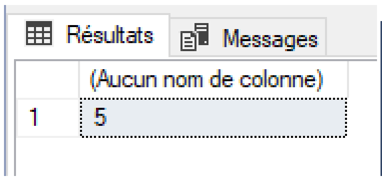

Résultats Messages

	limit
1	5

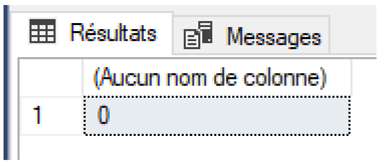

Résultats Messages

	limit
1	3

1st student:limit 2nd student: limit


Résultats Messages

	(Aucun nom de colonne)
1	5


Résultats Messages

	(Aucun nom de colonne)
1	0

1st student : count 2nd student : count

We have to change the inserted value to (3,555)and(3,666) because we already used (1,555) ,(1,666), (2,555)and(2,666) because we can not insert the same value multiple times. Otherwise we get a result as error like the previous task

TASK #10

Problem statement:

We need to reset our database by executing the script of example 5 again.

Solution:

We copy and execute again the script from the task 5.

Code:

```
use lab4
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)
```

Reasoning:

Result:

We recopy the script from task 5.

TASK #11

Problem statement:

We need to increase by using read committed isolation level and mutual locking.
If concurrent transactions lock only the record they need, mutual exclusion can be achieved.

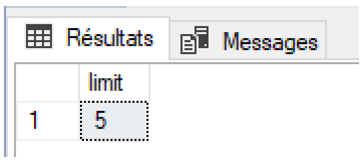
Solution:

We copy the following code :

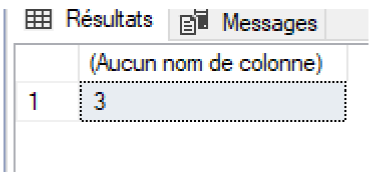
First signup	Second signup
(1) set transaction isolation level read committed begin transaction	(1) set transaction isolation level read committed begin transaction
(2) select limit from exam with(XLOCK) where id=1	(2) select limit from exam with(XLOCK) where id=1
(3) select count(*) from signup where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

Reasoning:

Result:



```
Résultats  Messages
   limit
1  5
```



```
Résultats  Messages
   (Aucun nom de colonne)
1  3
```

The transaction holds a read on the current row, and prevents other transactions from reading it.
Since the isolation level guarantees to read only committed data, it does not allow dirty read.

TASK #12

Problem statement:

We need to increase limit of the first exam to 6, then do again the task11.

Solution:

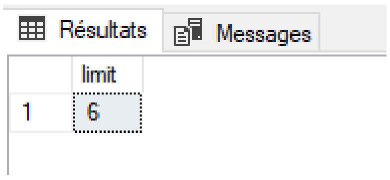
Code: increase limit of the first exam to 6.

```
use lab4
update exam set limit=6 where id=1
```

Recopy the code from the previous task.

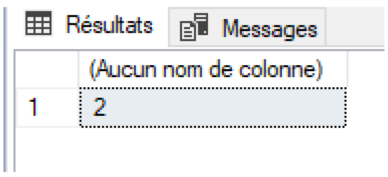
Reasoning:

Result:

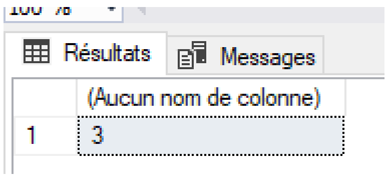


```
Résultats  Messages
   limit
1  6
```

get an icreased limit for the exams



```
Résultats  Messages
   (Aucun nom de colonne)
1  2
```



```
Résultats  Messages
   (Aucun nom de colonne)
1  3
```

Count from 1st signup   Count from 2nd signup

We have to change the inserted value to (2,333) because we already used (1,333) and we can not insert the same value multiple times.

TASK #13

Problem statement:

We need to do again the task 12 with a different method : the first student should signup to the first exam while the second one should signup to the second exam

Solution:

We copy the following code :

```
(6) set transaction isolation
    level read committed
    begin transaction

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

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

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

(10)      commit
```

First signup

```
(5) set transaction isolation
    level read committed
    begin transaction

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

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

(8) commit
```

Second signup

Reasoning:

Result:

The second student should signup to the second exam => change the id and the examid from 1 to 2.

We also have to change the inserted value to (3,333) because we already used (1,333) and(2,333) and we can not insert the same value multiple times.