E/15/385
Weerasinghe S.P.A.P.E.

Department of Computer Engineering

University of Peradeniya

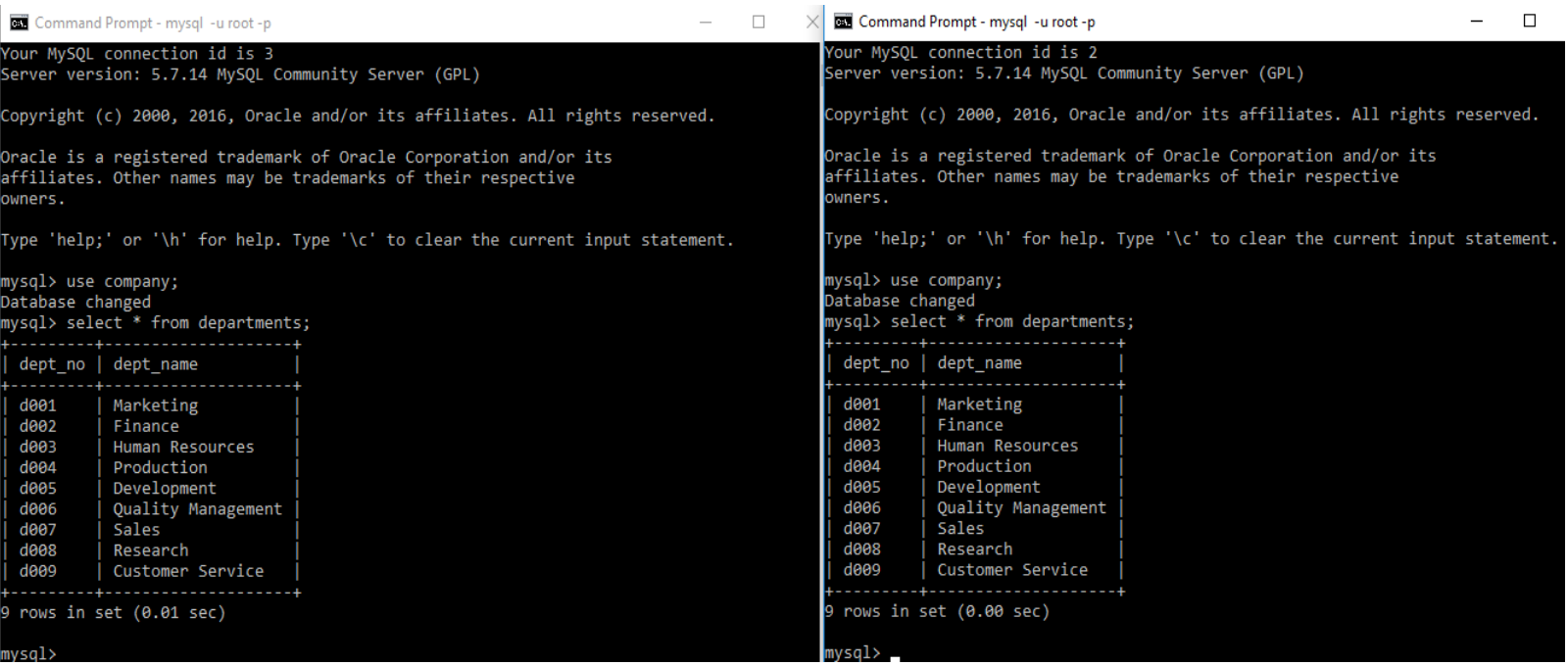
CO527 Advanced Database Systems

Lab Number : 04

Topic : Transaction Processing

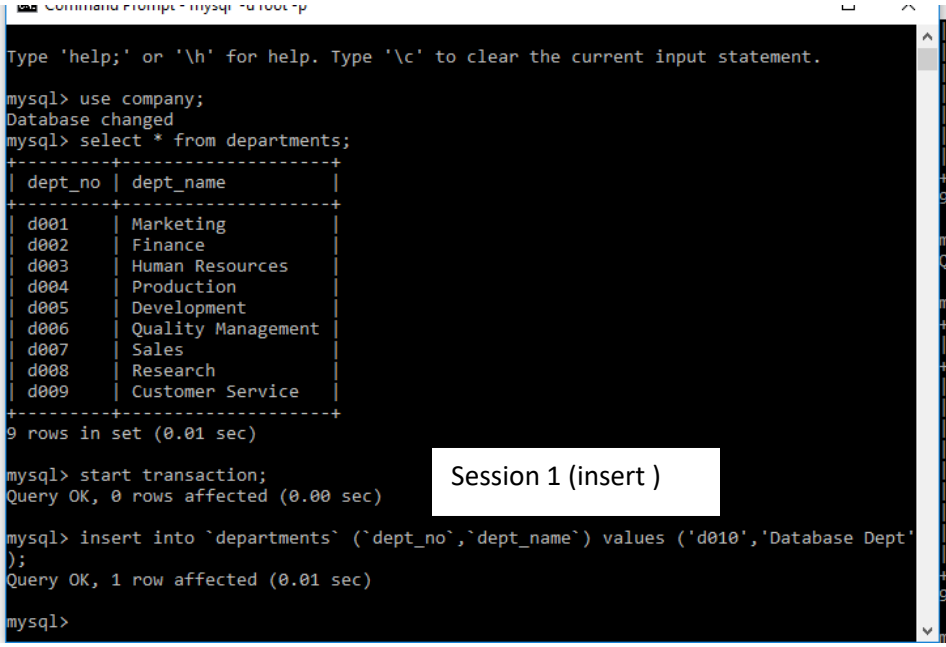
1. I of ACID

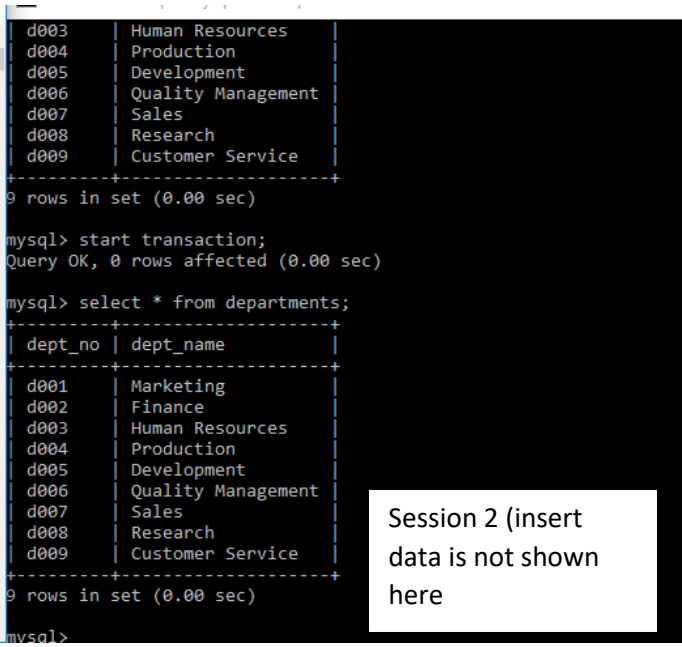
I. Issue a select query to view the current status of the departments table in both sessions.



II. Now, start transaction running start transaction in both sessions.

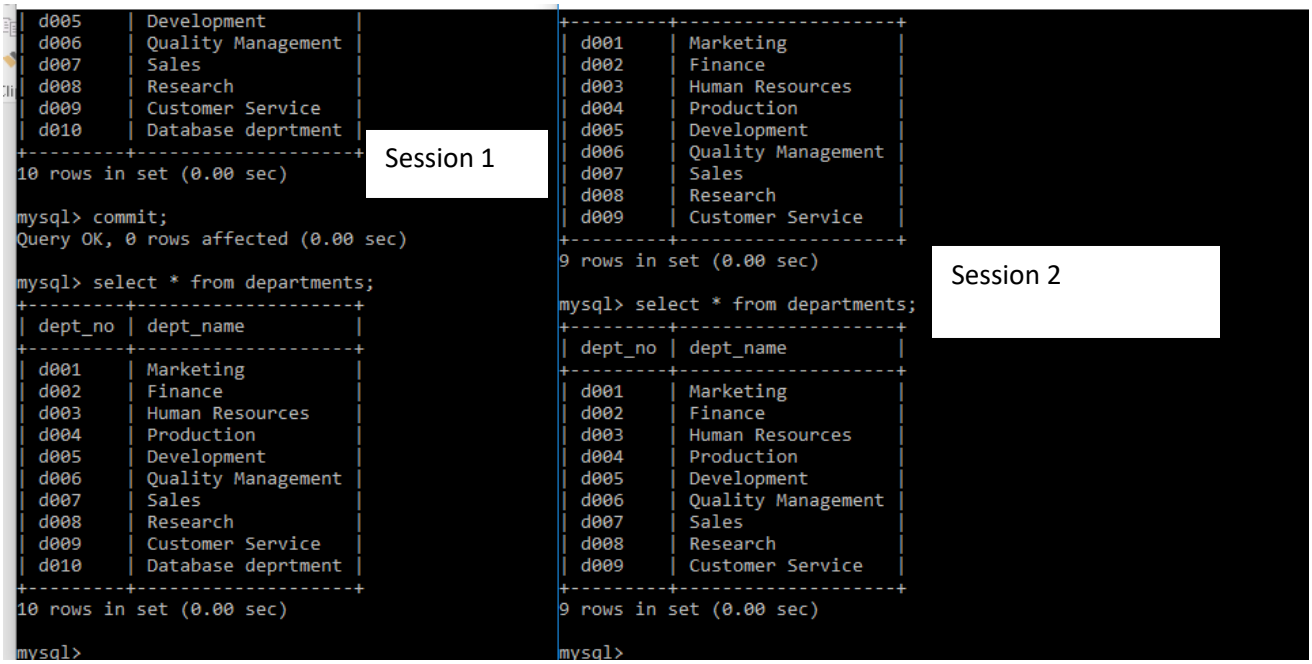
III. Insert a new row into the departments table from the 1st session and check if the changes are visible in the second session.





Insert data is only shown in the session 1.

IV. Commit changes in the 1st command window and check if you can see the updates done at 1st window in 2nd command window.




```
| d005    | Development        |
| d006    | Quality Management |
| d007    | Sales              |
| d008    | Research           |
| d009    | Customer Service   |
| d010    | Database deprtment |
+---------+--------------------+
10 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from departments;
+---------+--------------------+
| dept_no | dept_name          |
+---------+--------------------+
| d001    | Marketing          |
| d002    | Finance            |
| d003    | Human Resources    |
| d004    | Production         |
| d005    | Development        |
| d006    | Quality Management |
| d007    | Sales              |
| d008    | Research           |
| d009    | Customer Service   |
| d010    | Database deprtment |
+---------+--------------------+
10 rows in set (0.00 sec)

mysql>
```

```
+---------+--------------------+
| d001    | Marketing          |
| d002    | Finance            |
| d003    | Human Resources    |
| d004    | Production         |
| d005    | Development        |
| d006    | Quality Management |
| d007    | Sales              |
| d008    | Research           |
| d009    | Customer Service   |
+---------+--------------------+
9 rows in set (0.00 sec)

mysql> select * from departments;
+---------+--------------------+
| dept_no | dept_name          |
+---------+--------------------+
| d001    | Marketing          |
| d002    | Finance            |
| d003    | Human Resources    |
| d004    | Production         |
| d005    | Development        |
| d006    | Quality Management |
| d007    | Sales              |
| d008    | Research           |
| d009    | Customer Service   |
+---------+--------------------+
9 rows in set (0.00 sec)

mysql>
```

Even after commit from the session 1 data is not accessed to session 2.

V. Explain your observations before and after running the commit in the 1st window.

This explains the “ACID properties”.

Session 1 and session 2 are treated as two **isolated** transcaitons.

In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

Therefore session 1 and session 2 exists as they are the only transcation exists.

InnoDB offers all four transaction isolation levels described by the SQL:1992 standard: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE. The default isolation level for InnoDB is REPEATABLE READ.

Also **atomicity** too effects here.

This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

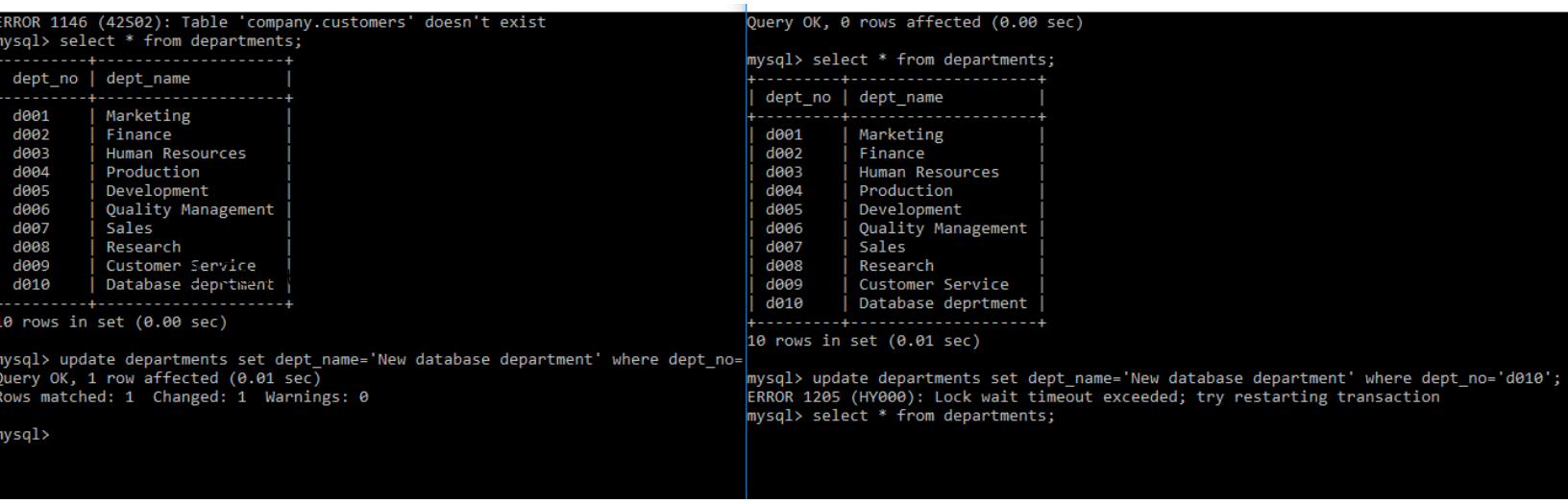
Therefore the states should be defined before the execution or after the execution.

Therefore due to above reasons isolation and atomicity we can't see the new data in the session 2.

2. Concurrent Updates

I. Try to do a concurrent update to the same row in departments table during two transactions

```
UPDATE departments SET dept_name = 'New database dept' WHERE dept_no
= 'd010';
```



```
ERROR 1146 (42S02): Table 'company.customers' doesn't exist
mysql> select * from departments;
+---------+--------------------+
| dept_no | dept_name          |
+---------+--------------------+
| d001    | Marketing          |
| d002    | Finance            |
| d003    | Human Resources    |
| d004    | Production         |
| d005    | Development        |
| d006    | Quality Management |
| d007    | Sales              |
| d008    | Research           |
| d009    | Customer Service   |
| d010    | Database deprtment |
+---------+--------------------+
10 rows in set (0.00 sec)

mysql> update departments set dept_name='New database department' where dept_no=
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> select * from departments;
+---------+--------------------+
| dept_no | dept_name          |
+---------+--------------------+
| d001    | Marketing          |
| d002    | Finance            |
| d003    | Human Resources    |
| d004    | Production         |
| d005    | Development        |
| d006    | Quality Management |
| d007    | Sales              |
| d008    | Research           |
| d009    | Customer Service   |
| d010    | Database deprtment |
+---------+--------------------+
10 rows in set (0.01 sec)

mysql> update departments set dept_name='New database department' where dept_no='d010';
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql> select * from departments;
```

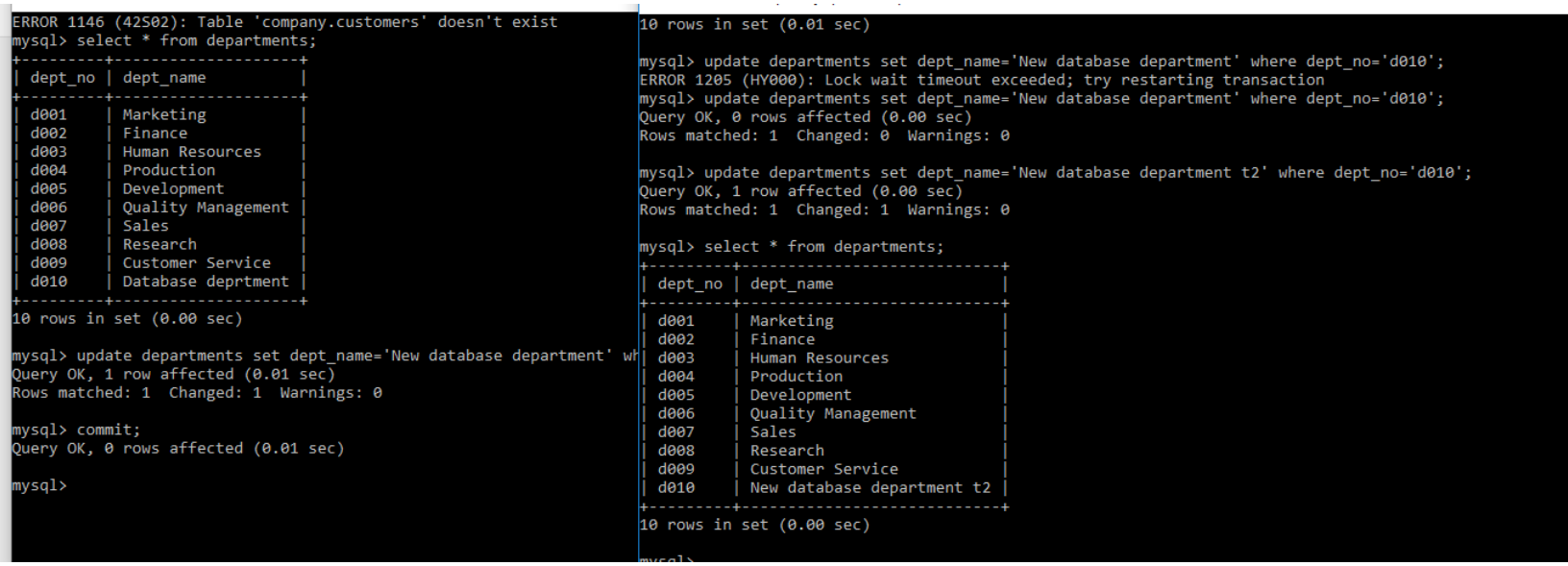
When the same row is tried to update on the transcation 2 , there was a error (exceeding time out error).

II. Explain what happens before ending any of the transactions.

Here the transaction 1 issues a write lock on tha row (because of update query). Before commiting that update od before releasing that write lock , we try to access and update the same row using transaction 2, but the transaction 2 cannot access it since the write lock of the transaction 1 is not yet released.Transcation wait sometime weather t1 realease locks

III. What happens when you commit your changes in the 1st session.

After commiting the transaction 1 we can update the same row from transaction 2.



Because commit release the write lock in the row from transaction 1.

What to Turn In

Use your imagination and words to write a scenario where using transactions is essential and then create the required tables and test how the transaction will effect your tables,

1. during the transaction execution.
2. after rollback statement
3. after the commit statement.
4. during 2 concurrent transactions, both of them update a record and both of them commit it.

Online banking system

Why we need transaction processing here?

Here All your money stored in banks is stored in the database, all your shares of DMAT account is stored in the database and many application constantly works on these data

In order to protect data and keep it consistent, any changes in this data need to be done in a transaction so that even in the case of failure data remain in the previous state before the start of a transaction.

I created two tables
1) Users
2) Account

Code

```
CREATE TABLE `users` (
  `User_name` varchar(100) NOT NULL,
  `Account_number` int(11) NOT NULL,
  `Branch` varchar(100) NOT NULL,
  `Birth date` date NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

//index for user table

ALTER TABLE `users`
  ADD PRIMARY KEY (`Account_number`);
```

```
//create Account table
CREATE TABLE `bank`.`Account` ( `Account_no` INT NOT NULL , `Balance` INT NOT NULL ,
`Interst_val` INT NOT NULL , PRIMARY KEY (`Account_no`)) ENGINE = MyISAM;
```

```
//insert into
INSERT INTO `account` (`Account_no`, `Balance`, `Interst_val`) VALUES
(300, 1000, 4),
(400, 5000, 4);



//index

ALTER TABLE `account`
  ADD PRIMARY KEY (`Account_no`);
```

Steps in a money transaction;

1) Verify account details.
2) Accept withdrawal request
3) Check balance
4) Update balance
4) Dispense money

A real world Scenario where we need the transaction processing.

Suppose your account balance is Rs.1000 and you make a withdrawal request of Rs.900. At fourth step, your balance is updated to Rs.900$ and Online Banking system stops working due to power outage.

Once power comes back and you again tried to withdraw money you surprised by seeing your balance just Rs.100 instead of Rs.1000. This is not acceptable by anybody.so we need a transaction to perform such task.

Reasons that can cause transaction failures are **due to power failure, system crash** etc.

To handle these types of problems we need to maintain "**ACID"** properties.And we need transaction processing in this kind of scenario.

Now lets explore what will happen

Scenario : Amila-A

Kanthi-B

B wants to send A 1000 rupees through this online banking system

Fisrt check the details required as in the following table.

```
Command Prompt - mysql  -u root -p
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database bank;
Query OK, 1 row affected (0.01 sec)

mysql> use bank;
Database changed
mysql> select * from account;
+------------+---------+-------------+
| Account_no | Balance | Interst_val |
+------------+---------+-------------+
|        300 |    1000 |           4 |
|        400 |    5000 |           4 |
+------------+---------+-------------+
2 rows in set (0.01 sec)

mysql> select * from users;
+-------------------+----------------+--------+------------+
| User_name         | Account_number | Branch | Birth date |
+-------------------+----------------+--------+------------+
| Amila Weerasinghe |            300 | Matale | 1996-01-21 |
| Kanthi Munasinghe |            400 | Kandy  | 1962-04-25 |
+-------------------+----------------+--------+------------+
2 rows in set (0.00 sec)
```

Here is the transaction

```
start transaction;
select balance from Account where Account_no='400';
select balance from Account where Account_no='300';
update Account set balance=balance-1000 where Account_no='400' ;
update Account set balance=balance+1000 where Account_no='300' ;
commit; //if all sql queries succed
```

rollback; //if any of Sql queries failed or error

savepoint sv_p1; //save point created so this use rollback in future cases.

```
mysql> start transaction
    -> select balance from Account where Account_no='400';
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to y
he right syntax to use near 'select balance from Account where Account_no='400'' at line 2
mysql> select balance from Account where Account_no='300';
+---------+
| balance |
+---------+
|    1000 |
+---------+
1 row in set (0.00 sec)

mysql> update Account set balance=balance-1000 where Account_no='400' ;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update Account set balance=balance+1000 where Account_no='300' ;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from account;
+------------+---------+-------------+
| Account_no | Balance | Interst_val |
+------------+---------+-------------+
|        300 |    2000 |           4 |
|        400 |    4000 |           4 |
+------------+---------+-------------+
2 rows in set (0.00 sec)

mysql>
```

The transaction has performed well.

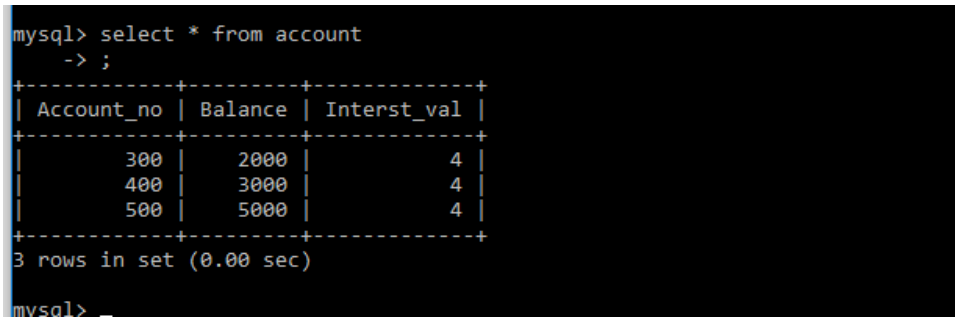
Use of this transaction processing ensure

Atomicity -All or nothing quality.

How tables are effected.

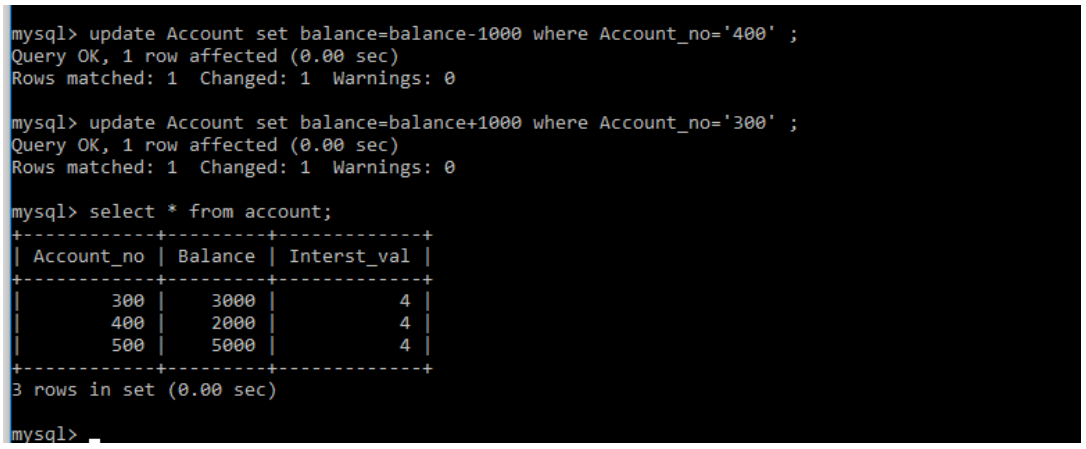
1. during the transaction execution..

Before the transaction.



```
mysql> select * from account
    -> ;
+------------+---------+-------------+
| Account_no | Balance | Interst_val |
+------------+---------+-------------+
|        300 |    2000 |           4 |
|        400 |    3000 |           4 |
|        500 |    5000 |           4 |
+------------+---------+-------------+
3 rows in set (0.00 sec)

mysql>
```

After transcation.



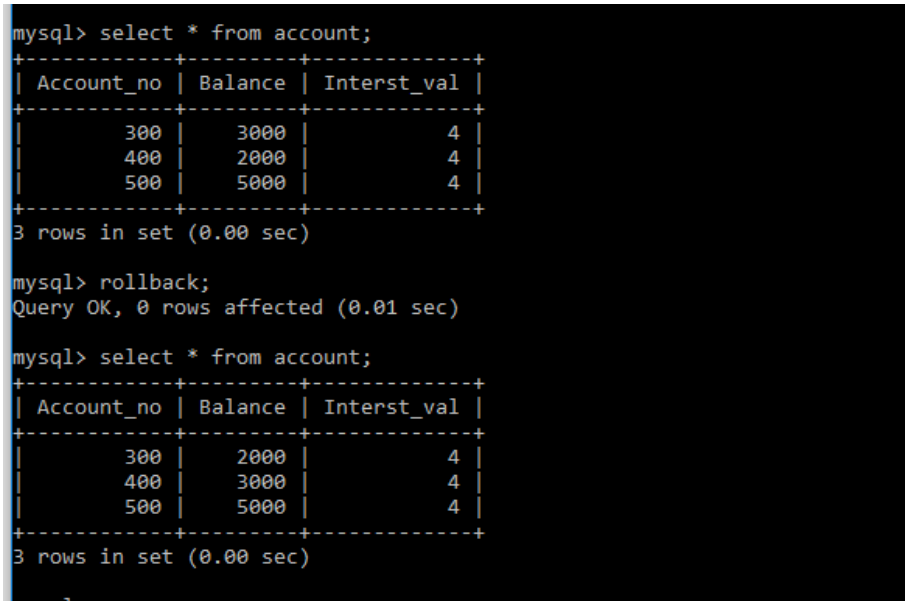
```
mysql> update Account set balance=balance-1000 where Account_no='400' ;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update Account set balance=balance+1000 where Account_no='300' ;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from account;
+------------+---------+-------------+
| Account_no | Balance | Interst_val |
+------------+---------+-------------+
|        300 |    3000 |           4 |
|        400 |    2000 |           4 |
|        500 |    5000 |           4 |
+------------+---------+-------------+
3 rows in set (0.00 sec)

mysql>
```

The balance of Account have changed successfully.(Rs.1000 has transferred successfully.)

We start the above with “Start transcation”.After use of this the particular locks are added to the rows which are to be updated.Also Start transcation gives isolation to the particular. Set of these commands are the transcation which ensures the above ACID properties.

2. after rollback statement.

When Rollback is called as soon after the above transaction.(before commit)



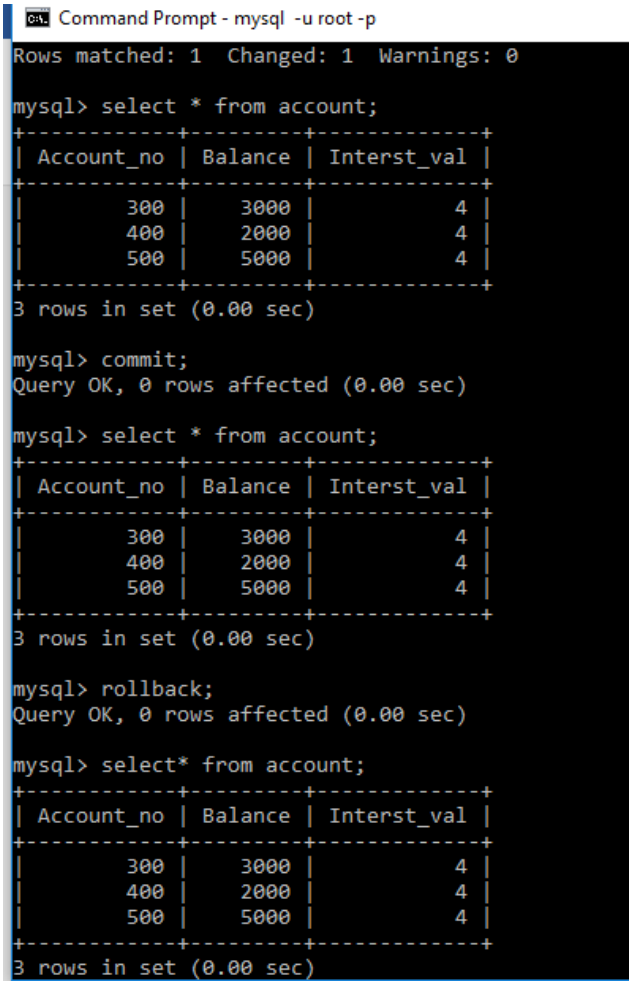
```
mysql> select * from account;
+------------+---------+-------------+
| Account_no | Balance | Interst_val |
+------------+---------+-------------+
|        300 |    3000 |           4 |
|        400 |    2000 |           4 |
|        500 |    5000 |           4 |
+------------+---------+-------------+
3 rows in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from account;
+------------+---------+-------------+
| Account_no | Balance | Interst_val |
+------------+---------+-------------+
|        300 |    2000 |           4 |
|        400 |    3000 |           4 |
|        500 |    5000 |           4 |
+------------+---------+-------------+
3 rows in set (0.00 sec)
```

The transferred amount have reversed.(table have changed to the state prior to transaction).

Here in the main transcation process I have added the “Rollback statement” to rollback to ensure that the database goes back to the previous consistent state if any of the queries failed or give error.
In case of system crash or power failure the transaction should be ROLLBACK to previous committed (consistent) state. Otherwise the amount of money will be errornous.

3. after the commit statement.

Executed transcation . Committed -> after that rollback



After a successful commit the rollback does not effect the tables.

The commit makes the changes which made in the buffer to database. So the changes become permanent. The updated values are saved to the database . Also after this the locks created by the transaction on certain data values are released.

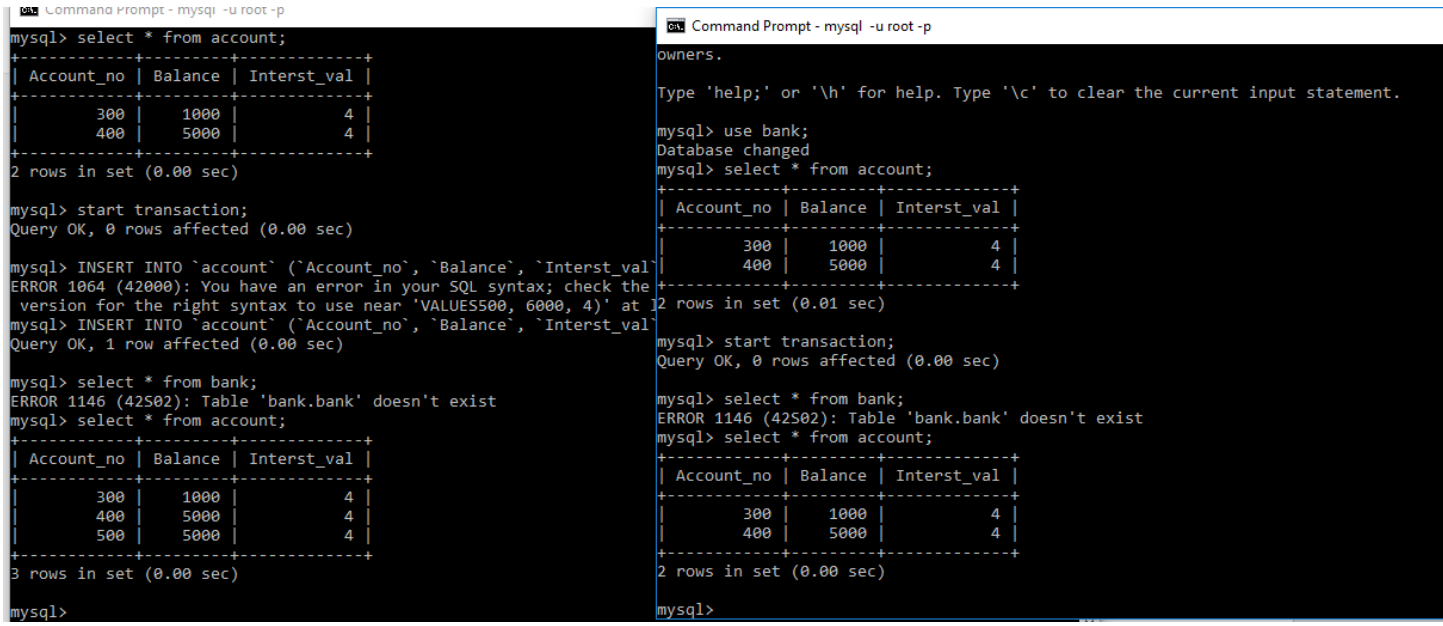
4. during 2 concurrent transactions, both of them update a record and both of them commit it.

Testing two concurrent transactions:

- Started two transactions in two command windows on the same database.(Using start transaction command)
- Inserted new row (INSERT INTO `account` (`Account_no`, `Balance`, `Interst_val`) VALUES(500, 5000, 4);)
- Checked weather I can access that newly inserted data in the other command windows. (select * from account).
- But cannot access it from that table.

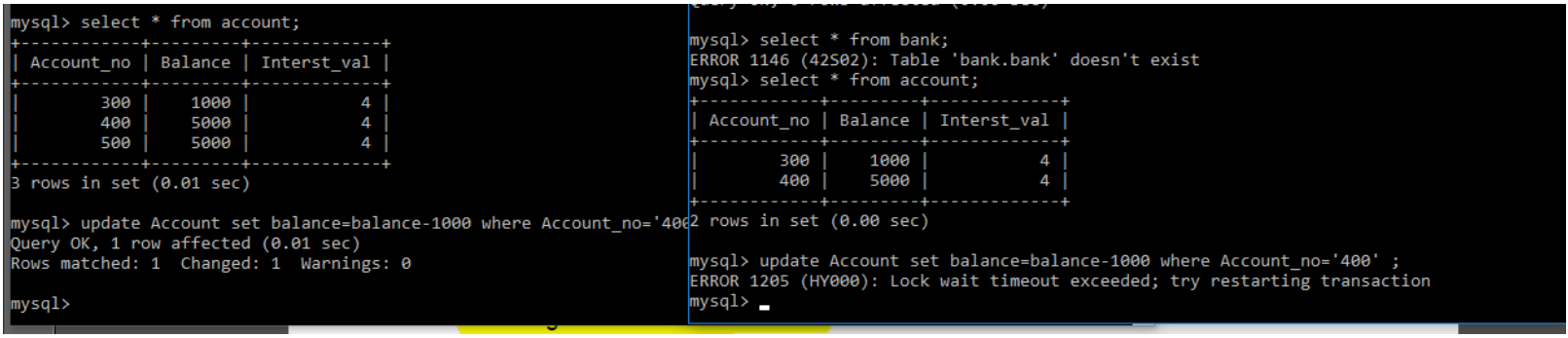
Explaination:

This happened due to the level of isolations. Transaction isolation is one of the foundations of database processing. Isolation is the I in the acronym ACID; the isolation level is the setting that fine-tunes the balance between performance and reliability, consistency, and reproducibility of results when multiple transactions are making changes and performing queries at the same time.
InnoDB offers all four transaction isolation levels described by the SQL:1992 standard: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE. The default isolation level for InnoDB is REPEATABLE READ.



Remarks: It is really important to be the tables Engine=InnoDB instead of other database Engine types.

Concurrent updates.



Here when two concurrent,

- Two trasactions start on the same Database.
- One transcation updates data(kanthi receives 1000 from Amila)
- The second transaction too tries to get another Rs.1000 from the same account.
- This is not happened.

Explaination:

A write lock is issued upon the particular data when updated by the first transaction. It cannot be updated by the second transcation as the write lock is not yet released.It gives timeout

```
|        500 |    5000 |           4 |
+------------+---------+-------------+
3 rows in set (0.01 sec)

mysql> update Account set balance=balance-1000 where Account_no='400';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from account;
+------------+---------+-------------+
| Account_no | Balance | Interst_val |
+------------+---------+-------------+
|        300 |    1000 |           4 |
|        400 |    4000 |           4 |
|        500 |    5000 |           4 |
+------------+---------+-------------+
3 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> update Account set balance=balance-1000 where Account_no='400' ;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql> select * from account;
+------------+---------+-------------+
| Account_no | Balance | Interst_val |
+------------+---------+-------------+
|        300 |    1000 |           4 |
|        400 |    5000 |           4 |
+------------+---------+-------------+
2 rows in set (0.00 sec)

mysql> select * from account;
+------------+---------+-------------+
| Account_no | Balance | Interst_val |
+------------+---------+-------------+
|        300 |    1000 |           4 |
|        400 |    5000 |           4 |
+------------+---------+-------------+
2 rows in set (0.00 sec)
```

When the first transaction is committed the second transcation can't see it .

This is due to Isolation property.

When both transaction committed.

```
mysql> select * from account;
+------------+---------+-------------+
| Account_no | Balance | Interst_val |
+------------+---------+-------------+
|        300 |    1000 |           4 |
|        400 |    4000 |           4 |
|        500 |    5000 |           4 |
+------------+---------+-------------+
3 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

```
mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+------------+---------+-------------+
| Account_no | Balance | Interst_val |
+------------+---------+-------------+
|        300 |    1000 |           4 |
|        400 |    4000 |           4 |
|        500 |    5000 |           4 |
+------------+---------+-------------+
3 rows in set (0.00 sec)

mysql>
```

Now the changes done on the both transactions and committed. When both transactions committed we can see the changes done by other transactions. because committing makes the transcations to end.