National University of Computer and Emerging Sciences

Lab Manual

"Transactions Manual"

Database Systems Lab

Spring 2022

Department of Computer Science

FAST-NU, Lahore, Pakistan

Objective
The purpose of this lab is to introduce the students to the transactions in sql.

Prerequisite
All the lab manuals till now.

Task Distribution

Total Time	
Introduction	
Exercise	

Transactions

In database terms, a **transaction** is any action that reads from and/or writes to a database. A transaction may consist of a simple SELECT statement to generate a list of table contents; it may consist of a series of related UPDATE statements to change the values of attributes in various tables; it may consist of a series of INSERT statements to add rows to one or more tables; or it may consist of a combination of SELECT, UPDATE, and INSERT statements. Each successful transaction leaves the database in a consistent state

Consistent State

A consistent state is one in which all the database integrity constraints are being fulfilled, which means that each transaction must be atomic i.e. it either runs completely or does not run at all.

A transaction is a logical unit of work which must be entirely executed or entirely aborted. Atomicity is the most important principal in the transactions.

Transaction Properties

Transactions possess the following properties that make the transactions trustworthy.

- Atomicity

- Consistency
- Isolation
- Durability
- Serializability

Atomicity

Requires all the operations of a transactions to be completed. If an instruction/query is not completely executed it should be aborted. A transaction is treated as a single indivisible logical unit. As mentioned above that the logical unit either executes completely or does not execute at all.

Consistency

Consistency dictates that each transaction from the database leaves the database in a consistent states i.e. no integrity constraints are violated. The transactions change the database from one consistent state to another consistent state.

Isolation

Isolation implies that the data used in one transaction can't be used in another transaction unless the first transaction is complete.

Durability

Durability ensures that once the changes are made in the database, they can't be undone even in the case of system failure.

Serializability

Serializability property belongs solely to the multiuser scenario it ensures that the schedule for the concurrent transactions yield a consistent result.

Transaction Isolation Levels

- Dirty read:

  The meaning of this term is as bad as it sounds. You're permitted to read uncommitted, or dirty data. You can achieve this effect by just opening an OS file that someone else is writing and reading whatever data happens to be there. Data integrity is compromised, foreign keys are violated, and unique constraints are ignored.

- Nonrepeatable read**:**

  This simply means that if you read a row at time T1 and try to reread that row at time T2, the row may have changed. It may have disappeared, it may have been updated, and so on.

- Phantom read:

This means that if you execute a query at time T1 and re-execute it at time T2, additional rows may have been added to the database, which may affect your results. This differs from a nonrepeatable read in that with a phantom read, data you already read hasn't been changed, but instead, more data satisfies your query criteria than before.

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
READ UNCOMMITTED	Permitted	Permitted	Permitted
READ COMMITTED	--	Permitted	Permitted
REPEATABLE READ	--	--	Permitted
SERIALIZABLE	--	--	--

- **Read committed** is an isolation level that guarantees that any data **read** was **committed** at the moment is **read**. It simply restricts the reader from seeing any intermediate, **uncommitted**, 'dirty' **read.**

- **Repeatable read** is a higher isolation level, that in addition to the guarantees of the **read** committed level, it also guarantees that any data **read** cannot change, if the transaction **reads** the same data again, it will find the previously **read** data in place, unchanged, and available to **read**.

- **SERIALIZABLE**
  Specifies the following:
  - Statements cannot read data that has been modified but not yet committed by other transactions.
  - No other transactions can modify data that has been read by the current transaction until the current transaction completes.
  - Other transactions cannot insert new rows with key values that would fall in the range of keys read by any statements in the current transaction until the current transaction completes.

Read committed example 1:

Session 1

```
begin tran
update emp set Salary=999 where ID=1
waitfor delay '00:00:15'
commit
```

Session 2

```
set transaction isolation level read committed
select Salary from Emp where ID=1
```

Read uncommitted example 1

Session 1

```
begin tran
update emp set Salary=999 where ID=1
waitfor delay '00:00:15'
rollback
```

Session 2

```
set transaction isolation level read uncommitted
select Salary from Emp where ID=1
```

Repeatable Read Example 1

Session 1

```
set transaction isolation level repeatable read
begin tran
select * from emp where ID in(1,2)
waitfor delay '00:00:15'
select * from Emp where ID in (1,2)
rollback
```

Session 2

```
update emp set Salary=999 where ID=1
```

Serializable Example 1

Assume table does not have index column.

Session 1

```
set transaction isolation level serializable
begin tran
select * from emp
waitfor delay '00:00:15'
select * from Emp
rollback
```

Session 2

```
insert into Emp(ID,Name,Salary)
values( 11,'Stewart',11000)
```

Types of Transactions

Unlike the other types of instructions and queries that are only of two types namely DDL and DML we have an additional type of transaction so the three types of transactions are as follows:

- DDL transactions
- DML transactions
- Transactions control statements

DML Transactions

Let us begin with the DML transactions.

Transaction Format

The format of transaction is as follows:

Begin Transaction

Sql queries (DDL or DML)

Commit/Rollback

A transaction can be either committed or rolled back

Commit

Commit command is used to save all the changes that are made in the sql queries in the transaction. The queries that follow the begin transaction command are a part of the transaction.

Rollback

Rollback takes the data back to the last save point or to the beginning of the transaction depending on the scenario, which means that all the changes since the last save-point are undone.

```
select * from students
```

100 %

Results | Messages

	StudentID	StudentName	StudentBatch	CGPA
1	1	Ali	2013	3.3
2	2	Aysha	2013	4
3	3	Ahmed	2013	2.2

```
begin transaction
insert into students (studentid, studentname, studentbatch, cgpa) values (4, 'Tanzeel',2009,3.8)
select * from students
```

100 %

Results | Messages

	StudentID	StudentName	StudentBatch	CGPA
1	1	Ali	2013	3.3
2	2	Aysha	2013	4
3	3	Ahmed	2013	2.2
4	4	Tanzeel	2009	3.8

```
rollback
select * from Students
```

100 %

Results | Messages

	StudentID	StudentName	StudentBatch	CGPA
1	1	Ali	2013	3.3
2	2	Aysha	2013	4
3	3	Ahmed	2013	2.2

Similarly, we can apply this to any of the DML queries. Before moving forward, let us see that every rollback must have a corresponding begin transaction. Because once a transaction has been committed it cannot be rolled back it is a permanent change in the database.

```
begin transaction
update Students set CGPA=4.0 where studentid=1
update Students set CGPA=3.0 where studentid=2
select * from Students
commit
 rollback
```

100 %

Messages

```
Msg 3903, Level 16, State 1, Line 21
The ROLLBACK TRANSACTION request has no corresponding BEGIN TRANSACTION.
```

Transactions in Stored Procedures:

In the following code you will see “save transaction” in the code this is used to create a save point in the transaction it is a point in the transaction where in case of roll back the transaction should roll back to i.e. the changes will be undone till this point in the transaction.

See https://docs.microsoft.com/en-us/sql/t-sql/functions/trancount-transact-sql?view=sql-server-2017

For **@@trancount**

```
create procedure trans_in_proc
(
@course_id int,
@credit_hour varchar(30)
)
as begin
begin transaction
save transaction savepoint;

begin try
update courses set coursecredithours=@credit_hour where (courseid = @course_id);
end try
begin catch
if @@trancount>0
begin
rollback transaction savepoint;
end
end catch
commit transaction
end;
```

Messages

```
Command(s) completed successfully.
```

```
select * from courses
```

Results | Messages

	CourseID	CourseName	CourseCreditHours	InstructorID
1	1	Computer Programming	3	1
2	2	Computer Organization	3	2
3	3	Computer Programming Lab	1	NULL

```
exec trans_in_proc @course_id=3,@credit_hour=2

select * from courses
```

Results | Messages

	CourseID	CourseName	CourseCreditHours	InstructorID
1	1	Computer Programming	3	1
2	2	Computer Organization	3	2
3	3	Computer Programming Lab	2	NULL

```
exec trans_in_proc @course_id=8,@credit_hour=2


select * from courses
```

100 %

Results | Messages

	CourseID	CourseName	CourseCreditHours	InstructorID
1	1	Computer Programming	3	1
2	2	Computer Organization	3	2
3	3	Computer Programming Lab	2	NULL

Transactions with/without Locks:

Transaction with no locks don't allow multiple command s to be run on the same data set however in case of transactions with no locks allow you to run multiple commands on the same data set.

Note:

In case of nested transactions, the rollback takes back to the outer most transactions.