



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

**Batch: A2**

**Roll No.: 1911027**

**Experiment / assignment / tutorial No. 8**

**Grade: AA / AB / BB / BC / CC / CD / DD**

**Title: Implementing TCL/DCL**

**Objective:** To be able to Implement TCL and DCL.

**Expected Outcome of Experiment:**

CO 2: Convert entity-relationship diagrams into relational tables, populate a relational database and formulate SQL queries on the data Use SQL for creation and query the database.

CO 4: Demonstrate the concept of transaction, concurrency control and recovery techniques.

**Books/ Journals/ Websites referred:**

1. Dr. P.S. Deshpande, SQL and PL/SQL for Oracle 10g.Black book, Dreamtech Press
2. www.db-book.com
3. Korth, Silberchatz, Sudarshan : “Database Systems Concept”, 5<sup>th</sup> Edition , McGraw Hill
4. Elmasri and Navathe,”Fundamentals of database Systems”, 4<sup>th</sup> Edition,PEARSON Education.

**Resources used:** PostgreSQL

**Theory**

DCL stands for Data Control Language.

DCL is used to control user access in a database.

This command is related to the security issues.



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

Using DCL command, it allows or restricts the user from accessing data in database schema.

DCL commands are as follows,

**GRANT**

**REVOKE**

It is used to grant or revoke access permissions from any database user.

**GRANT command** gives user's access privileges to the database.

This command allows specified users to perform specific tasks.

**Syntax:**

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE |
REFERENCES | TRIGGER }
      [, ...] | ALL [ PRIVILEGES ] }
ON { [ TABLE ] table_name [, ...]
     | ALL TABLES IN SCHEMA schema_name [, ...] }
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
OPTION ]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name
[, ...] )
      [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
ON [ TABLE ] table_name [, ...]
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
OPTION ]
```

**Example**

```
GRANT INSERT ON films TO PUBLIC;
GRANT ALL PRIVILEGES ON kinds TO ram;
GRANT admins TO krishna;
```

**REVOKE command** is used to cancel previously granted or denied permissions.

This command withdraw access privileges given with the GRANT command.

It takes back permissions from user.

**Syntax:**

```
REVOKE [ GRANT OPTION FOR ]
      { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE |
REFERENCES | TRIGGER }
      [, ...] | ALL [ PRIVILEGES ] }
ON { [ TABLE ] table_name [, ...]
```



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

```
        | ALL TABLES IN SCHEMA schema_name [, ...] }
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [,
... ] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
ON [ TABLE ] table_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { USAGE | SELECT | UPDATE }
    [, ...] | ALL [ PRIVILEGES ] }
ON { SEQUENCE sequence_name [, ...]
    | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]
```

### Example

```
REVOKE INSERT ON films FROM PUBLIC;
REVOKE ALL PRIVILEGES ON kinds FROM Madhav;
REVOKE admins FROM Keshav;
```

TCL stands for **Transaction Control Language**.

This command is used to manage the changes made by DML statements.

TCL allows the statements to be grouped together into logical transactions.

**TCL commands are as follows:**

1. COMMIT
2. SAVEPOINT
3. ROLLBACK
4. SET TRANSACTION

**COMMIT command** saves all the work done. It ends the current transaction and makes permanent changes during the transaction

### Syntax:

```
commit;
```



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

**SAVEPOINT command** is used for saving all the current point in the processing of a transaction. It marks and saves the current point in the processing of a transaction. It is used to temporarily save a transaction, so that you can rollback to that point whenever necessary.

Syntax

```
SAVEPOINT savepoint_name
```

**ROLLBACK** command restores database to original since the last COMMIT. It is used to restores the database to last committed state.

Syntax:

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ]  
savepoint_name
```

Example

```
BEGIN;  
    INSERT INTO table1 VALUES (1);  
    SAVEPOINT my_savepoint;  
    INSERT INTO table1 VALUES (2);  
    ROLLBACK TO SAVEPOINT my_savepoint;  
    INSERT INTO table1 VALUES (3);  
COMMIT;
```

The above transaction will insert the values 1 and 3, but not 2.

**SET TRANSACTION** is used for placing a name on a transaction. You can specify a transaction to be read only or read write. This command is used to initiate a database transaction.

Syntax:

```
SET TRANSACTION [Read Write | Read Only];
```

The SET TRANSACTION command sets the characteristics of the current transaction. It has no effect on any subsequent transactions. SET SESSION CHARACTERISTICS sets the default transaction characteristics for subsequent transactions of a session. These defaults can be overridden by SET TRANSACTION for an individual transaction.

The available transaction characteristics are the transaction isolation level, the transaction access mode (read/write or read-only), and the deferrable mode. In addition,



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

a snapshot can be selected, though only for the current transaction, not as a session default.

The isolation level of a transaction determines what data the transaction can see when other transactions are running concurrently:

### **READ COMMITTED**

A statement can only see rows committed before it began. This is the default.

### **REPEATABLE READ**

All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction.

### **SERIALIZABLE**

All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction. If a pattern of reads and writes among concurrent serializable transactions would create a situation which could not have occurred for any serial (one-at-a-time) execution of those transactions, one of them will be rolled back with a `serialization_failure` error.

### **Examples**

With the default read committed isolation level.

```
process A: BEGIN; -- the default is READ COMMITTED

process A: SELECT sum(value) FROM purchases;
--- process A sees that the sum is 1600

process B: INSERT INTO purchases (value) VALUES (400)
--- process B inserts a new row into the table while
--- process A's transaction is in progress

process A: SELECT sum(value) FROM purchases;
--- process A sees that the sum is 2000

process A: COMMIT;
```

If we want to avoid the changing sum value in process A during the lifespan of the transaction, we can use the repeatable read transaction mode.

```
process A: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
process A: SELECT sum(value) FROM purchases;
--- process A sees that the sum is 1600

process B: INSERT INTO purchases (value) VALUES (400)
--- process B inserts a new row into the table while
--- process A's transaction is in progress
```



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

```
process A: SELECT sum(value) FROM purchases;  
--- process A still sees that the sum is 1600  
  
process A: COMMIT;
```

The transaction in process A will freeze its snapshot of the data and offer consistent values during the life of the transaction.

Repeatable reads are not more expensive than the default read commit transaction. There is no need to worry about performance penalties. However, applications must be prepared to retry transactions due to serialization failures.

Let's observe an issue that can occur while using the repeatable read isolation level — the `could not serialize access due to concurrent update` error.

```
process A: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;  
process B: BEGIN;  
process B: UPDATE purchases SET value = 500 WHERE id = 1;  
process A: UPDATE purchases SET value = 600 WHERE id = 1;  
-- process A wants to update the value while process B is changing it  
-- process A is blocked until process B commits  
  
process B: COMMIT;  
process A: ERROR: could not serialize access due to concurrent update  
-- process A immediately errors out when process B commits
```

If process B would roll back, then its changes are negated and repeatable read can proceed without issues. However, if process B commits the changes then the repeatable read transaction will be rolled back with the error message because it can not modify or lock the rows changed by other processes after the repeatable read transaction has begun.

demonstrate the differences between the two isolation modes.

```
process A: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;  
process A: SELECT sum(value) FROM purchases;  
process A: INSERT INTO purchases (value) VALUES (100);  
process B: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;  
process B: SELECT sum(value) FROM purchases;  
process B: INSERT INTO purchases (id, value);  
process B: COMMIT;  
process A: COMMIT;
```

With Repeatable Reads everything works, but if we run the same thing with a Serializable isolation mode, process A will error out.



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

```
process A: BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
process A: SELECT sum(value) FROM purchases;
process A: INSERT INTO purchases (value) VALUES (100);
process B: BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
process B: SELECT sum(value) FROM purchases;
process B: INSERT INTO purchases (id, value);
process B: COMMIT;
process A: COMMIT;
ERROR: could not serialize access due to read/write
dependencies among transactions
DETAIL: Reason code: Canceled on identification as
a pivot, during commit attempt.
HINT: The transaction might succeed if retried.
```

Both transactions have modified what the other transaction would have read in the select statements. If both would allow to commit this would violate the Serializable behaviour, because if they were run one at a time, one of the transactions would have seen the new record inserted by the other transaction.

### **Implementation Screenshots (Problem Statement, Query and Screenshots of Results):**

**Problem Statement:** Online pharmacy management system where user can order medicines from a particular pharmacy by comparing prices from different pharmacies available. And also checking availability of medicines in the pharmacy.

### **DCL: Data Control Language:**

1) This query will grant all permissions to specified user on pharmacy table.

#### **Query:**

Creating a user:

```
CREATE USER PharmacyOwner@localhost IDENTIFIED BY 'Nayan@123';
```



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

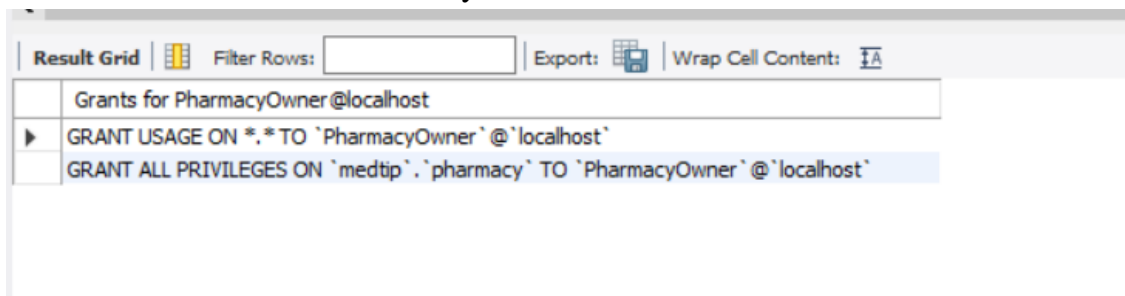
Granting privileges:

GRANT ALL ON pharmacy TO PharmacyOwner@localhost;

**Screenshot:**

Showing privileges:

SHOW GRANTS FOR PharmacyOwner@localhost;



Result Grid	Filter Rows:	Export:	Wrap Cell Content:
Grants for PharmacyOwner@localhost			
▶	GRANT USAGE ON *.* TO `PharmacyOwner` @ `localhost`		
	GRANT ALL PRIVILEGES ON `medtip`, `pharmacy` TO `PharmacyOwner` @ `localhost`		

**2) This query will revoke select permission on pharmacy table from specified user.**

**Query:**

REVOKE SELECT ON pharmacy FROM PharmacyOwner@localhost;

**Screenshot:**

Showing grants:

SHOW GRANTS FOR PharmacyOwner@localhost;





**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

Grants for PharmacyOwner@localhost
GRANT USAGE ON *.* TO 'PharmacyOwner'@'localhost'
GRANT INSERT, UPDATE, DELETE, CREATE, DROP, REFERENCES, INDEX, ALTER, CREATE VIEW, SHOW VIEW, TRIGGER ON 'medip', 'pharmacy' TO 'PharmacyOwner'@'localhost'

## TCL: Transaction Control Language:

### 1) Query:

#### Connection 1 Before Commit:

SET autocommit=0;

insert into pharmacy values(16,"Daisy Pharma","Bandra",400099,"LT Road");

SELECT \* FROM pharmacy;

Result Grid | Filter Rows: | Edit: | Export:

	Pharma_ID	Pharma_Name	Area	Pin	street
▶	12	Kritarth Pharma	Ghatkopar	400086	MG Road
	13	Hussein Pharma	Ghatkopar	400016	PG Road
	14	Nayan Pharma	Ghatkopar	400077	KG Road
	15	Manan Pharma	Ghatkopar	400077	LG Road
	16	Daisy Pharma	Bandra	400099	LT Road
*	NULL	NULL	NULL	NULL	NULL

#### Connection 2 Before Commit:

SELECT \* from pharmacy;



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

	Pharma_ID	Pharma_Name	Area	Pin	street
▶	12	Kritarth Pharma	Ghatkopar	400086	MG Road
	13	Hussein Pharma	Ghatkopar	400016	PG Road
	14	Nayan Pharma	Ghatkopar	400077	KG Road
	15	Manan Pharma	Ghatkopar	400077	LG Road
•	NULL	NULL	NULL	NULL	NULL

**Screenshot:**

**Connection 1 After Commit:**

COMMIT ;

**Connection 2 After Commit:**

SELECT \* from pharmacy;

	Pharma_ID	Pharma_Name	Area	Pin	street
▶	12	Kritarth Pharma	Ghatkopar	400086	MG Road
	13	Hussein Pharma	Ghatkopar	400016	PG Road
	14	Nayan Pharma	Ghatkopar	400077	KG Road
	15	Manan Pharma	Ghatkopar	400077	LG Road
	16	Daisy Pharma	Bandra	400099	LT Road
•	NULL	NULL	NULL	NULL	NULL

pharmacy 2 x

**2) Query:**

insert into customer values (7,"Daisy","Daniel","Disti",1,"PM Road",112133,"Kurla");

SELECT \* FROM customer;



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

**Before ROLLBACK:**

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

FA

	Cust_ID	C_FName	C_MName	C_LName	Gender	Street	Pin	Area
▶	1	Kri	Tar	Th	1	MG Road	400086	Ghatkopar
	2	Na	Ya	N	1	KG Road	400077	Ghatkopar
	3	Hus	Sei	N	1	PG Road	400033	Ghatkopar
	4	Pi	Na	Ki	0	KL Road	401033	Ghatk
	5	K	an	K	0	RS Road	111033	Ghatkop
	6	Kri	Tar	Th	0	RS Road	111033	Ghatkop
	7	Daisy	Daniel	Disti	1	PM Road	112133	Kurla
✱	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

customer 2 ✕

**ROLLBACK query: ROLLBACK ;**

**Screenshot:**

**After ROLLBACK;**

Cust_ID	C_FName	C_MName	C_LName	Gender	Street	Pin	Area
1	Kri	Tar	Th	1	MG Road	400086	Ghatkopar
2	Na	Ya	N	1	KG Road	400077	Ghatkopar
3	Hus	Sei	N	1	PG Road	400033	Ghatkopar
4	Pi	Na	Ki	0	KL Road	401033	Ghatk
5	K	an	K	0	RS Road	111033	Ghatkop
6	Kri	Tar	Th	0	RS Road	111033	Ghatkop
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

**3) Query:**

```
INSERT INTO medicine VALUES(7,"Benadryl","intel",20,8,"Cough Syrup","Manan Pharma","2024-12-12",15,1);
```

```
INSERT INTO medicine VALUES(8,"Phospha","AWS",50,9,"Heart Problem","Kritarth Pharma","2024-12-12",12,1);
```



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

```
INSERT INTO medicine VALUES(9,"CoughSyr","kin",60,11,"Cough  
Syrup","Hussein Pharma","2024-12-12",13,1);
```

```
SAVEPOINT savep1;
```

```
SELECT * FROM medicine;
```

	Med_ID	Med_Name	Med_Company	Quantity	Price	Med_Type	Pharma_Name	Exp_Date	Phar_ID	Cus_ID
▶	1	Crocin	intel	200	15	headache	Kritarth Pharma	2022-12-12	12	1
	2	Paracetamol	AMD	30	100	Antibiotic	HusseinPharma	2023-12-12	13	4
	3	Crocin	intel	200	10	headache	Nayan Pharma	2024-12-12	14	1
	4	Crocin	intel	200	8	headache	Manan Pharma	2024-12-12	15	1
	5	ALDOX	intel	200	8	headache	Manan Pharma	2024-12-12	15	1
	7	Benadryl	intel	20	8	Cough Syrup	Manan Pharma	2024-12-12	15	1
	8	Phospha	AWS	50	9	Heart Problem	Kritarth Pharma	2024-12-12	12	1
	9	CoughSyr	kin	60	11	Cough Syrup	Hussein Pharma	2024-12-12	13	1
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
INSERT INTO medicine  
VALUES(10,"Moxcha","alwar",70,14,"Dengue","Manan Pharma","2024-12-  
12",15,1);
```

```
INSERT INTO medicine  
VALUES(11,"Septimide","Keras",270,15,"Malaria","Kritarth Pharma","2024-  
12-12",12,1);
```

```
INSERT INTO medicine  
VALUES(12,"Tetramine","metar",230,17,"Pneumonia","Hussein  
Pharma","2024-12-12",13,1);
```

```
SAVEPOINT savep2 ;
```

```
SELECT * FROM medicine;
```



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

Result Grid										
Filter Rows: [ ] Edit: [ ] Export/Import: [ ] Wrap Cell Content: [ ]										
	Med_ID	Med_Name	Med_Company	Quantity	Price	Med_Type	Pharma_Name	Exp_Date	Phar_ID	Cus_ID
▶	1	Crocine	intel	200	15	headache	Kritarth Pharma	2022-12-12	12	1
	2	Paracetamol	AMD	30	100	Antibiotic	HusseinPharma	2023-12-12	13	4
	3	Crocine	intel	200	10	headache	Nayan Pharma	2024-12-12	14	1
	4	Crocine	intel	200	8	headache	Manan Pharma	2024-12-12	15	1
	5	ALDOX	intel	200	8	headache	Manan Pharma	2024-12-12	15	1
	7	Benadryl	intel	20	8	Cough Syrup	Manan Pharma	2024-12-12	15	1
	8	Phospha	AWS	50	9	Heart Problem	Kritarth Pharma	2024-12-12	12	1
	9	CoughSyr	kin	60	11	Cough Syrup	Hussein Pharma	2024-12-12	13	1
	10	Moxcha	alwar	70	14	Dengue	Manan Pharma	2024-12-12	15	1
	11	Septimide	Keras	270	15	Malaria	Kritarth Pharma	2024-12-12	12	1
	12	Tetramine	metar	230	17	Pneumonia	Hussein Pharma	2024-12-12	13	1
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

**Screenshot:**

ROLLBACK to SAVEPOINT savep1;  
SELECT \* FROM medicine;

Result Grid										
Filter Rows: [ ] Edit: [ ] Export/Import: [ ] Wrap Cell Content: [ ]										
	Med_ID	Med_Name	Med_Company	Quantity	Price	Med_Type	Pharma_Name	Exp_Date	Phar_ID	Cus_ID
▶	1	Crocine	intel	200	15	headache	Kritarth Pharma	2022-12-12	12	1
	2	Paracetamol	AMD	30	100	Antibiotic	HusseinPharma	2023-12-12	13	4
	3	Crocine	intel	200	10	headache	Nayan Pharma	2024-12-12	14	1
	4	Crocine	intel	200	8	headache	Manan Pharma	2024-12-12	15	1
	5	ALDOX	intel	200	8	headache	Manan Pharma	2024-12-12	15	1
	7	Benadryl	intel	20	8	Cough Syrup	Manan Pharma	2024-12-12	15	1
	8	Phospha	AWS	50	9	Heart Problem	Kritarth Pharma	2024-12-12	12	1
	9	CoughSyr	kin	60	11	Cough Syrup	Hussein Pharma	2024-12-12	13	1
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

**Conclusion:** By performing this experiment commands related to data control language and transaction control language is been understood. Also executed some queries on our created database.

**Postlab question:**

**1. Discuss ACID properties of transaction with suitable example.**

**ANS)** A transaction is a set of related changes which is used to achieve some of the ACID properties. Transactions are tools to achieve the ACID properties. A transaction can be defined as a group of tasks. A single task is



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

the minimum processing unit which cannot be divided further. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability – commonly known as ACID properties – in order to ensure accuracy, completeness, and data integrity. Let's take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

**A's Account:**

Open Account(A)

Old.Balance=A.balance

New.Balance=Old.Balance – 500

A.balance= New.Balance

Close.Account(A).

**B's Account:**

Open Account(B)

Old.Balance=B.balance

New.Balance=Old.Balance + 500

B.balance= New.Balance

Close.Account(B).

The ACID Properties of a Transaction are as follows:

**1) Atomicity:**

- This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none.





**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

- 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.
- When an update occurs to a database, either all or none of the update becomes available to anyone beyond the user or application performing the update.
- This update to the database is called a transaction and it either commits or aborts.
- This means that only a fragment of the update cannot be placed into the database, should a problem occur with either the hardware or the software involved.

## **2) Consistency:**

- Consistency is the ACID property that ensures that any changes to values in an instance are consistent with changes to other values in the same instance.
- The database must remain in a consistent state after any transaction.
- No transaction should have any adverse effect on the data residing in the database.
- If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- For example, we may have a table or a record on which two transaction are trying to read or write at the same time.
- Careful mechanisms are created in order to prevent mismanagement of these sharable resources, so that there should not be any change in the way a transaction performs.
- A transaction which deposits Rs 100/- to account A must deposit the same amount whether it is acting alone or in conjunction with another transaction that may be trying to deposit or withdraw some amount at the same time.



**K. J. Somaiya College of Engineering, Mumbai-77**  
(Autonomous College Affiliated to University of Mumbai)

### **3) Isolation:**

- 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.
- For example, user A withdraws \$100 and user B withdraws \$250 from user Z's account, which has a balance of \$1000.
- Since both A and B draw from Z's account, one of the users is required to wait until the other user transaction is completed, avoiding inconsistent data.
- Now, B can withdraw 250 from this \$900 balance.

### **4) Durability:**

- The database should be durable enough to hold all its latest updates even if the system fails or restarts.
- If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data.
- If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- Features to consider for durability:
  - Recovery to the most recent successful commit after a database software failure.
  - Recovery to the most recent successful commit after an application software failure.
  - Recovery to the most recent successful commit after a CPU failure.
  - Recovery to the most recent successful backup after a disk failure.