**Promt : Give in detailed documentation of TCL command and syntax. Explain in detail with 2 examples as a beginer in PostgreSQL**

**TCL Commands**

**TCL stands for Transaction Control Language. These commands are used to manage transactions within a database. A transaction is a sequence of one or more SQL statements that are executed as a single, indivisible unit of work. TCL commands are crucial for ensuring data integrity and consistency.**

The primary TCL commands are:

* **BEGIN**: Starts a new transaction block. All subsequent SQL statements will be part of this transaction until it is ended by COMMIT or ROLLBACK.
* **COMMIT**: Saves all changes made in a transaction to the database permanently.
* **ROLLBACK**: Undoes all changes made in a transaction, reverting the database to its state before the transaction began.
* **SAVEPOINT**: Creates a point within a transaction to which you can later roll back.

**Syntax**

The syntax for TCL commands is simple and straightforward.

Apologies for that omission. You are correct, the BEGIN command is an essential part of Transaction Control Language (TCL) in PostgreSQL, as it explicitly starts a new transaction block. I will provide a corrected and more detailed explanation.

**BEGIN Command**

The **BEGIN** command, also known as START TRANSACTION, is used to explicitly start a new transaction block. In PostgreSQL, by default, every SQL statement runs in its own transaction (this is called autocommit mode). The BEGIN command allows you to group multiple SQL statements together so they are treated as a single, atomic unit of work. This means either all of the statements succeed and are permanently saved with COMMIT, or if any of them fail, all of the changes are undone with ROLLBACK.

**TCL Commands in Detail**

**TCL (Transaction Control Language)** commands are used to manage these transactions, ensuring data integrity and consistency. The primary TCL commands are:

* **BEGIN**: Starts a new transaction block. All subsequent SQL statements will be part of this transaction until it is ended by COMMIT or ROLLBACK.
* **COMMIT**: Saves all changes made within the current transaction to the database permanently.
* **ROLLBACK**: Undoes all changes made within the current transaction, reverting the database to its state before the BEGIN command was issued.
* **SAVEPOINT**: Creates a temporary marker within a transaction, allowing you to ROLLBACK to a specific point without undoing the entire transaction.

**Syntax**

* BEGIN:

SQL

BEGIN;

* COMMIT:

SQL

COMMIT;

* ROLLBACK:

SQL

ROLLBACK;

* SAVEPOINT:

SQL

SAVEPOINT savepoint\_name;

* + savepoint\_name: A unique identifier for the savepoint.
* ROLLBACK TO SAVEPOINT:

SQL

ROLLBACK TO savepoint\_name;

* + **savepoint\_name**: The name of the specific savepoint you want to roll back to.

**Examples for Beginners**

Let's assume we have a simple accounts table with a starting balance for two users.

SQL

-- Assume this table and data already exist

CREATE TABLE accounts (

account\_id INT PRIMARY KEY,

account\_holder VARCHAR(50),

balance NUMERIC(10, 2)

);

INSERT INTO accounts VALUES

(1, 'Alice', 1000.00),

(2, 'Bob', 500.00);

**Example 1: Using COMMIT and ROLLBACK for a transfer 💰**

A classic example of a transaction is transferring money between accounts. A successful transfer requires two steps: debiting one account and crediting another. If one step fails, the entire transaction should be canceled to prevent data inconsistency.

SQL

BEGIN; -- Start a new transaction

-- Step 1: Debit Alice's account

UPDATE accounts

SET balance = balance - 200.00

WHERE account\_id = 1;

-- Step 2: Credit Bob's account

UPDATE accounts

SET balance = balance + 200.00

WHERE account\_id = 2;

-- Let's assume both updates are successful, so we commit the changes.

COMMIT;

If, for some reason, the second UPDATE statement had failed (e.g., due to a system error), we would have used **ROLLBACK;** instead of COMMIT;. This would have undone both updates, leaving Alice's and Bob's balances unchanged.

**Example 2: Using SAVEPOINT for a partial rollback 🔄**

SAVEPOINT is useful when you want to perform a series of operations but need the option to undo some, but not all, of them.

SQL

BEGIN; -- Start a new transaction

-- First, update Alice's account

UPDATE accounts SET balance = balance - 50.00 WHERE account\_id = 1;

-- Create a savepoint here

SAVEPOINT first\_transfer;

-- Now, perform another update (e.g., transfer to a third account)

UPDATE accounts SET balance = balance + 50.00 WHERE account\_id = 3;

-- Let's say the second update fails or is incorrect, so we roll back to the savepoint.

ROLLBACK TO first\_transfer;

-- We can now commit the first, successful part of the transaction.

COMMIT;

In this example, the update to account\_id = 1 is preserved because we rolled back to the first\_transfer savepoint, which was created after that update. The second update, which occurred after the savepoint, is undone. The COMMIT at the end then finalizes only the changes up to the savepoint.

**⚠️ Common Mistakes to Avoid**

1. **Forgetting to COMMIT**:  
   Your changes stay in transaction memory and are not saved permanently.
2. **Running DDL commands inside transactions (like CREATE TABLE)**:  
   Not all PostgreSQL DDLs can be rolled back once committed.
3. **Using transactions in auto-commit mode (default in some environments)**:  
   In tools like pgAdmin or DBeaver, each query may auto-commit unless you turn off that setting.