Promt : Give in detailed documentation of truncate command and syntax. Explain in detail with 2 examples as a beginner in postgreSQL

**TRUNCATE COMMAND**

The TRUNCATE command in PostgreSQL is used to quickly remove all rows from a set of tables. Unlike DELETE, it doesn't scan the table or generate individual row-level logs, making it significantly faster, especially for large tables. TRUNCATE also resets the table's identity columns or sequences.

**Syntax**

The basic syntax for the TRUNCATE command is as follows:

SQL

TRUNCATE [ TABLE ] table\_name [, ... ]

[ RESTART IDENTITY | CONTINUE IDENTITY ]

[ CASCADE | RESTRICT ]

* **TRUNCATE [ TABLE ] table\_name [, ... ]**: The TRUNCATE keyword is followed by the name of the table (or a comma-separated list of tables) that you want to empty. The TABLE keyword is optional.
* **RESTART IDENTITY**: This is an optional clause. When specified, it automatically resets any sequences associated with the table's identity columns to their starting value. For example, if you have an id column that auto-increments, this option will reset the counter so the next inserted row will have id = 1. This is the default behavior if not specified.
* **CONTINUE IDENTITY**: Also optional, this clause is the opposite of RESTART IDENTITY. It tells PostgreSQL to keep the sequence counters as they are. The next inserted row will use the next available number in the sequence, continuing from where it left off.
* **CASCADE**: This option automatically truncates all tables that have foreign-key references to the table(s) being truncated. It's a powerful and potentially dangerous option, as it can clear data from multiple related tables with a single command.
* **RESTRICT**: This is the default behavior. If a table has a foreign key reference from another table, TRUNCATE will fail and an error will be returned, preventing accidental data loss. This is the safer option.

**Key Differences from DELETE**

| Feature | TRUNCATE | DELETE |
| --- | --- | --- |
| **Speed** | Extremely fast, as it deallocates the table's data pages | Slower, as it deletes rows one by one |
| **Logging** | Minimal logging; logs the deallocation of data pages | Generates a log entry for each deleted row |
| **Transactions** | Is transactional and can be rolled back | Is transactional and can be rolled back |
| **WHERE Clause** | Cannot be used with a WHERE clause | Can be used with a WHERE clause |
| **IDENTITY Columns** | Can reset identity columns with RESTART IDENTITY | Doesn't affect sequence counters by default |
| **Foreign Keys** | Requires CASCADE to work with foreign keys | Works with foreign keys by default (depending on the constraint) |

**Examples**

**Example 1: Basic Truncate**

Let's say you have a table called sales\_data that stores daily sales records. At the end of the month, you want to clear this table to start fresh.

SQL

-- Assume 'sales\_data' is an existing table

-- This command will remove all rows from 'sales\_data'

-- It will also reset any auto-incrementing identity columns

TRUNCATE TABLE sales\_data;

**Explanation:** This is the most common use case. The command removes all data from the sales\_data table. If the table has an id column that auto-increments, the next row inserted will have an id of 1.

**Example 2: Truncate with Foreign Keys and Restarting Identity**

Imagine two tables: users and user\_sessions. The user\_sessions table has a foreign key referencing the users table. We want to clear both tables and restart their identity sequences.

SQL

-- First, let's create the tables to illustrate the example

CREATE TABLE users (

user\_id INT GENERATED ALWAYS AS IDENTITY,

username VARCHAR(50) );

CREATE TABLE user\_sessions (

session\_id INT GENERATED ALWAYS AS IDENTITY,

user\_id INT REFERENCES users(user\_id) ON DELETE CASCADE,

session\_start TIMESTAMP );

-- Insert some sample data

INSERT INTO users (username) VALUES ('Alice'), ('Bob');

INSERT INTO user\_sessions (user\_id, session\_start) VALUES (1, NOW()), (2, NOW());

-- Now, truncate both tables

-- The CASCADE keyword is crucial here because 'user\_sessions' references 'users'

TRUNCATE TABLE users, user\_sessions RESTART IDENTITY CASCADE;

**Explanation:** In this example, we use TRUNCATE on two tables at once: users and user\_sessions.

* **users, user\_sessions**: Specifies the two tables to be truncated.
* **RESTART IDENTITY**: This ensures that after the tables are cleared, the identity sequences for both user\_id and session\_id will be reset to 1.
* **CASCADE**: This is essential. Without CASCADE, the command would fail with an error because user\_sessions has a foreign key constraint referencing users. The CASCADE option tells PostgreSQL to not only truncate users but also all tables that depend on it, which in this case is user\_sessions.

**Key Points About TRUNCATE**

* **Faster than DELETE** for large tables because it does not scan rows individually.
* **Cannot use WHERE** — it’s all or nothing.
* **Releases disk space** immediately (unless inside a transaction).
* **Cannot be used if the table is referenced by a foreign key** (unless using CASCADE).

**When to Use TRUNCATE**

* When you need to **quickly remove all rows** from a table.
* When you don’t care about triggers firing.
* When you want to reset auto-increment IDs.
* For **temporary** or **staging tables** that you refresh often.

**When NOT to Use TRUNCATE**

* When you need to delete only **specific rows** (use DELETE ... WHERE instead).
* When you need triggers to execute upon deletion.
* When foreign key constraints prevent truncating (unless you use CASCADE carefully).