**Prompt : Provide me Brief information about Delete query in postgrade Sql and provide me syntax and examples and also explain me where we can use delete query.**

**The DELETE Command in PostgreSQL**

The DELETE command is a fundamental SQL statement used to remove one or more existing rows from a table. It is a **Data Manipulation Language (DML)** command, meaning it changes the data in your database.

Basic Theory:

The primary function of the DELETE command is to permanently remove data. The most important part of this command is the WHERE clause, which specifies exactly which rows to delete. Without a WHERE clause, the DELETE command will remove all rows from the table, so it must be used with extreme caution.

**Basic Syntax and Examples**

The basic DELETE statement consists of the DELETE FROM and the optional WHERE clauses.

**Syntax:**

SQL

DELETE FROM table\_name

WHERE condition;

1. Deleting Specific Rows:

This is the most common and safest use of the DELETE command. You use a WHERE clause to specify which rows to remove.

* **Example:**

SQL

DELETE FROM employees

WHERE employee\_id = 101;

*This command removes the single row where the employee\_id is 101.*

2. Deleting All Rows:

If you omit the WHERE clause, the DELETE command will remove every single row from the table.

* **Example:**

SQL

DELETE FROM employees;

***Warning:*** *This command is destructive and irreversible without a backup. For large tables, it can also be very slow. For a faster alternative to remove all data, consider using the TRUNCATE command, which is a DDL command and performs differently.*

**Key Clauses Used with DELETE**

* **FROM**: Specifies the table from which you want to remove data.
* **WHERE**: The crucial filter that determines which rows to delete.
* **USING**: A PostgreSQL-specific clause that allows you to join other tables to the DELETE command, enabling you to delete rows based on a condition in a different table.
* **RETURNING**: Another PostgreSQL-specific clause that returns the data of the rows that were just deleted. This is useful for auditing or logging the changes.

**Where the DELETE Command is Used**

The DELETE command is a vital tool for data management. It's used in several contexts beyond simple row removal.

1. **Primary Data Removal:** The most obvious use. You use DELETE to remove obsolete records, incorrect entries, or data that is no longer needed.
2. Using DELETE with the USING Clause (Advanced):

This is a powerful way to delete records from one table that are related to data in another table. It's a safer alternative to using subqueries for this purpose.

* + **Example:**

SQL

-- Delete all employees who work in a department that has been marked as inactive.

DELETE FROM employees

USING departments

WHERE employees.department\_id = departments.id

AND departments.status = 'inactive';

-- Delete all products that have not been ordered in the last year.

DELETE FROM products

WHERE product\_id NOT IN (

SELECT product\_id FROM orders WHERE order\_date > NOW() - INTERVAL '1 year'

);

1. Cascading Deletes (ON DELETE CASCADE):

This is an automated feature defined in a table's foreign key constraint. When a row in the parent table is deleted, all related rows in the child table are automatically deleted as well.

* + **Example:**

SQL

-- If an employee is deleted, their related address record is automatically deleted.

CREATE TABLE addresses (

...

employee\_id INT REFERENCES employees(id) ON DELETE CASCADE

);

*In this case, running DELETE FROM employees WHERE id = 10; would also automatically trigger a DELETE on the addresses table for that employee.*

**DELETE in Functions, Triggers, and Subqueries**

The DELETE command is not just a standalone query for manual data removal. It is a powerful tool used within the logic of other database objects to automate data cleanup, enforce data integrity, and perform dynamic deletions.

**1. DELETE in Functions**

A **function** is a reusable block of code that encapsulates business logic. Using DELETE inside a function allows you to centralize and automate data removal tasks, which can be called easily from your application or other parts of your database.

* **Theory & Use:**
  + **Automated Cleanup:** You can write a function to periodically delete old or obsolete records based on specific criteria.
  + **Centralized Logic:** If a specific deletion process involves multiple steps or conditions, you can encapsulate it in a single function call.
* **Example: A function to archive old log entries**

SQL

CREATE FUNCTION delete\_old\_logs(p\_age\_in\_days INT)

RETURNS INT AS $$

DECLARE

deleted\_count INT;

BEGIN

-- The DELETE command is the core of this function.

DELETE FROM application\_logs

WHERE created\_at < NOW() - (p\_age\_in\_days || ' days')::INTERVAL

RETURNING \*; -- We use RETURNING to count the deleted rows.

GET DIAGNOSTICS deleted\_count = ROW\_COUNT;

RETURN deleted\_count;

END;

$$ LANGUAGE plpgsql;

-- How to use the function to delete logs older than 90 days:

SELECT delete\_old\_logs(90);

**2. DELETE in Triggers**

A **trigger** is a rule that automatically executes a function when a specific database event occurs (INSERT, UPDATE, or DELETE). DELETE is often used within a trigger function to maintain data consistency by removing related records.

* **Theory & Use:**
  + **Cascading Deletes:** A trigger on a "parent" table can automatically delete records from a "child" table. While PostgreSQL's ON DELETE CASCADE is often preferred for simple relationships, a trigger allows for more complex, conditional cascading.
  + **Cleanup:** An AFTER DELETE trigger can clean up temporary data or related records in an unrelated table.
* Example: An AFTER DELETE trigger to clean up related data

Let's say a product has related reviews. When a product is deleted, you want to clean up its reviews.

SQL

-- 1. Create a function that deletes related reviews

CREATE FUNCTION cleanup\_reviews() RETURNS TRIGGER AS $$

BEGIN

-- The DELETE command removes reviews related to the deleted product.

DELETE FROM reviews

WHERE product\_id = OLD.product\_id;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

-- 2. Create the trigger to fire the function after a product is deleted

CREATE TRIGGER after\_product\_delete

AFTER DELETE ON products

FOR EACH ROW

EXECUTE FUNCTION cleanup\_reviews();

**3. DELETE with Subqueries**

A **subquery** is a SELECT statement nested within another query. Using a subquery in the WHERE clause of a DELETE statement allows you to dynamically determine which rows to delete based on a complex condition from another table.

* **Theory & Use:**
  + **Dynamic Deletion:** The subquery generates a list of values (e.g., IDs) that the outer DELETE query uses to filter the rows to be removed.
  + **Complex Conditions:** Subqueries are ideal when the deletion condition depends on a group of records, an aggregate function, or data from a table that isn't directly joinable.
* **Example: Delete users who have been inactive for a long time**

SQL

-- Delete all users who have not placed an order in the last 180 days.

DELETE FROM users

WHERE user\_id NOT IN (

-- The subquery returns a list of user IDs who have placed recent orders.

SELECT DISTINCT user\_id

FROM orders

WHERE order\_date > NOW() - INTERVAL '180 days'

);

*In this example, the subquery dynamically finds the active users, and the outer DELETE command removes everyone who is not on that list.*

**Important Note:** The DELETE command is a transactional operation. This means you can wrap it in a BEGIN; and COMMIT; block to safely test your command. If you make a mistake, you can use ROLLBACK; to undo the changes.