**Prompt : - Provide a detailed document for INSERT command in PostgreSQL with theoretical overview, complete syntax, constraints and their interaction with insert, 2 clear examples and best practices for beginners.**

The INSERT command in PostgreSQL is used to add new rows of data into a table. It's a fundamental Data Manipulation Language (DML) command for populating databases.

**Theoretical Overview**

When you execute an INSERT command, you're telling the database to create a new record, or row, and fill its columns with specific values. The values you provide must match the data types of the columns in the table. If you don't specify a value for a column, PostgreSQL will either use a default value if one is defined, or it will insert a NULL value. This command can insert a single row, multiple rows, or the results of a query.

**Complete Syntax**

The basic syntax for the INSERT command is:

SQL

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...)

[ON CONFLICT conflict\_action];

* **INSERT INTO table\_name**: Specifies the table where the data will be inserted.
* **(column1, column2, ...)**: An optional list of columns to insert data into. If omitted, you must provide values for all columns in the order they were defined in the table.
* **VALUES (value1, value2, ...)**: The values to be inserted, corresponding to the columns specified. You can insert multiple rows by separating value sets with commas: VALUES (value1a, value2a), (value1b, value2b).
* **[ON CONFLICT conflict\_action]**: This optional clause is used to handle conflicts that occur during an insert, such as a violation of a UNIQUE or PRIMARY KEY constraint. The conflict\_action can be DO NOTHING or DO UPDATE SET .... This is a powerful feature for upsert (update or insert) operations.

**Constraints and their Interaction with INSERT**

Constraints are rules enforced on data in a table. They play a crucial role in the behavior of the INSERT command.

* **PRIMARY KEY**: Ensures that the value in the specified column(s) is unique and not null. If you try to insert a row with a duplicate primary key, the INSERT command will fail unless an ON CONFLICT clause is used.
* **UNIQUE**: Similar to a primary key, but a table can have multiple unique constraints. It ensures that all values in a column or set of columns are unique. Inserting a duplicate value will raise an error.
* **NOT NULL**: Prevents a column from having a NULL value. If you don't provide a value for a NOT NULL column and no default is set, the INSERT will fail.
* **FOREIGN KEY**: Enforces referential integrity. It requires that the value in a column must exist in a column of another table. If you try to insert a foreign key value that doesn't exist in the referenced table, the INSERT will fail.
* **CHECK**: A CHECK constraint ensures that all values in a column satisfy a specific condition. For example, a CHECK constraint might ensure that a price is always greater than zero. If the inserted value violates this condition, the INSERT will be rejected.

**2 Clear Examples**

Let's assume we have a table named products with the following structure:

SQL

CREATE TABLE products (

product\_id SERIAL PRIMARY KEY,

product\_name VARCHAR(100) NOT NULL,

price NUMERIC(10, 2) CHECK (price > 0),

stock\_quantity INTEGER DEFAULT 0 );

**Example 1: Basic Single-Row Insert**

This example inserts a single product, providing values for the product\_name and price columns. The product\_id will be automatically generated by the SERIAL type, and stock\_quantity will use its DEFAULT value.

SQL

INSERT INTO products (product\_name, price)

VALUES ('Laptop', 1200.50);

**Example 2: Inserting Multiple Rows**

This example demonstrates how to insert multiple products in a single command, which is more efficient than running multiple separate INSERT statements.

SQL

INSERT INTO products (product\_name, price, stock\_quantity)

VALUES

('Mouse', 25.00, 50),

('Keyboard', 75.25, 30),

('Monitor', 350.00, 15);

**Best Practices for Beginners**

* **Always Specify Column Names**: It's a good practice to explicitly list the columns you're inserting into, as shown in the examples. This makes your code more readable and resilient to changes in table structure (e.g., if a new column is added later).
* **Use VALUES for Multiple Rows**: When inserting multiple rows, combine them into a single INSERT statement using multiple VALUES clauses. This is much faster and reduces the overhead of connecting to the database for each insert.
* **Respect Data Types**: Ensure the values you provide match the data types of the columns. Inserting a string into an integer column will result in an error. Use single quotes for strings and dates ('text', '2025-08-07').
* **Be Mindful of Constraints**: Before inserting, be aware of the constraints on the table. Forgetting about NOT NULL or UNIQUE constraints is a common mistake for beginners.
* **Understand SERIAL and DEFAULT**: Columns with SERIAL types or DEFAULT values don't need to be included in the column list, as the database will handle their values automatically.
* **Use RETURNING**: If you need to retrieve the values of the inserted row, such as the auto-generated product\_id, you can add RETURNING \*; or RETURNING product\_id; to the end of your INSERT statement.