

# #PostgreSQL#



## Section 13. PostgreSQL Constraints

Primary key – illustrate how to define a primary key when creating a table or adding a primary key to an existing table.

Foreign key – show you how to define foreign key constraints when creating a new table or adding foreign key constraints for existing tables.

DELETE CASCADE – show you how to automatically delete rows in child tables when the corresponding rows in the parent table are deleted.

CHECK constraint – add logic to check value based on a Boolean expression.

UNIQUE constraint – ensure that values in a column or a group of columns are unique across the table.

NOT NULL constraint – ensure values in a column are not NULL.

DEFAULT constraint – specify a default value for a column using the DEFAULT constraint.

Can You Teach Me above all the topics with example and show output also(create a simple table and insert some values, Also Explain clear cut explanation and Explain why this needed )



Sure! Let's go through each of the PostgreSQL constraints with examples, output, and explanations:

## 1. Primary Key

A primary key is a unique identifier for a row in a table. It cannot contain `NULL` values and must be unique across the table. It is used to ensure that each row in the table is unique.

### Example:

```
sql

-- Create a table with a primary key
CREATE TABLE users (
  user_id SERIAL PRIMARY KEY, -- SERIAL for auto incrementing unique IDs
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL UNIQUE
);

-- Insert data into the table
INSERT INTO users (username, email) VALUES ('john_doe', 'john@example.com');
INSERT INTO users (username, email) VALUES ('jane_doe', 'jane@example.com');
```

### Output:

```
sql
```

user_id	username	email
1	john_doe	john@example.com
2	jane_doe	jane@example.com

## Explanation:

- **Primary Key (`user\_id SERIAL PRIMARY KEY`):** The `user\_id` column is auto-incremented, ensuring each user has a unique identifier.
- **Why is it needed?:** The primary key ensures that each row in the table can be uniquely identified, which is critical for referencing and linking tables in a relational database.

## 2. Foreign Key

A foreign key is used to link two tables together. It ensures that a column or a group of columns in a table matches the primary key of another table, enforcing referential integrity.

## Example:

```
sql
```

```
-- Create two tables, one as the parent (products) and one as the child (order_items)
CREATE TABLE products (
  product_id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  price DECIMAL(10, 2) NOT NULL
);

CREATE TABLE order_items (
  order_item_id SERIAL PRIMARY KEY,
  order_id INT NOT NULL,
  product_id INT REFERENCES products(product_id), -- Foreign key references products table
  quantity INT NOT NULL
);

-- Insert data into products
INSERT INTO products (name, price) VALUES ('Product A', 100.00);
INSERT INTO products (name, price) VALUES ('Product B', 200.00);

-- Insert data into order_items
INSERT INTO order_items (order_id, product_id, quantity) VALUES (1, 1, 2); -- Valid
INSERT INTO order_items (order_id, product_id, quantity) VALUES (2, 2, 3); -- Valid
INSERT INTO order_items (order_id, product_id, quantity) VALUES (3, 3, 1); -- Invalid (product_id 3 does not exist)
```

## Output:

```
diff
```

order_item_id	order_id	product_id	quantity
1	1	1	2
2	2	2	3

## Explanation:

- **Foreign Key** (`product\_id` INT REFERENCES products(product\_id)): The `product\_id` in `order\_items` must exist in the `products` table, ensuring referential integrity.
- **Why is it needed?:** It maintains a relationship between `products` and `order\_items`, preventing orphaned records and preserving data integrity.

## 3. DELETE CASCADE

The **DELETE CASCADE** constraint automatically deletes rows in child tables when the corresponding rows in the parent table are deleted.

## Example:

```
sql

-- Create a parent table (categories) and a child table (products)
CREATE TABLE categories (
  category_id SERIAL PRIMARY KEY,
  name VARCHAR(50) NOT NULL
);

CREATE TABLE products (
  product_id SERIAL PRIMARY KEY,
  category_id INT REFERENCES categories(category_id) ON DELETE CASCADE,
  name VARCHAR(100) NOT NULL,
  price DECIMAL(10, 2) NOT NULL
);

-- Insert data into categories
INSERT INTO categories (name) VALUES ('Electronics');
INSERT INTO categories (name) VALUES ('Furniture');

-- Insert data into products
INSERT INTO products (category_id, name, price) VALUES (1, 'Smartphone', 299.99);
INSERT INTO products (category_id, name, price) VALUES (2, 'Desk', 149.99);

-- Delete a category
DELETE FROM categories WHERE category_id = 1;
```

## Output:

```
diff

product_id | category_id | name      | price
-----|-----|-----|-----
2          | 2          | Desk      | 149.99
```

## Explanation:

- **DELETE CASCADE** (`ON DELETE CASCADE`): When a row in `categories` is deleted, all associated rows in `products` with the same `category\_id` are also deleted.
- **Why is it needed?:** It automatically removes related child records when a parent record is deleted, preventing orphaned records and keeping the database consistent.

## 4. CHECK Constraint

A `CHECK` constraint allows you to add a logical condition to a column, restricting the values that can be inserted into it based on a Boolean expression.

### Example:

```
sql

-- Create a table with a CHECK constraint
CREATE TABLE employees (
  employee_id SERIAL PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  age INT CHECK (age > 0 AND age < 100), -- Age must be between 1 and 99
  department VARCHAR(50) NOT NULL
);

-- Insert data into the table
INSERT INTO employees (name, age, department) VALUES ('Alice', 30, 'HR'); -- Valid
INSERT INTO employees (name, age, department) VALUES ('Bob', -5, 'Finance'); -- Invalid (age
must be > 0)
```

### Output:

```
diff

employee_id | name  | age | department
-----|-----|-----|-----
1          | Alice | 30  | HR
```

### Explanation:

- **CHECK (`age > 0 AND age < 100`):** Ensures that the `age` column only accepts values between 1 and 99.
- **Why is it needed?:** It prevents invalid data from being inserted into the table, maintaining data integrity.

## 5. UNIQUE Constraint

The `UNIQUE` constraint ensures that all values in a column (or combination of columns) are unique across the table. It helps prevent duplicate data.

### Example:

```
sql

-- Create a table with a UNIQUE constraint
CREATE TABLE customers (
  customer_id SERIAL PRIMARY KEY,
  email VARCHAR(100) NOT NULL UNIQUE,
  phone VARCHAR(15) NOT NULL
);
```

```
-- Insert data into the table
INSERT INTO customers (email, phone) VALUES ('customer1@example.com', '1234567890'); -- Valid
INSERT INTO customers (email, phone) VALUES ('customer1@example.com', '9876543210'); -- Invalid (email is not unique)
```

## Output:

sql

customer_id	email	phone
1	customer1@example.com	1234567890

## Explanation:

- **UNIQUE** (`email VARCHAR(100) NOT NULL UNIQUE`): Ensures that all email addresses are unique.
- **Why is it needed?:** It prevents duplicate email addresses, which could cause data inconsistency.

## 6. NOT NULL Constraint

The **NOT NULL** constraint ensures that a column cannot have **NULL** values. It requires data to be entered for that column.

## Example:

sql

```
-- Create a table with a NOT NULL constraint
CREATE TABLE employees (
  employee_id SERIAL PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  age INT,
  department VARCHAR(50) NOT NULL
);

-- Insert data into the table
INSERT INTO employees (name, age, department) VALUES ('Alice', 30, 'HR'); -- Valid
INSERT INTO employees (name, age) VALUES ('Bob', NULL, 'Finance'); -- Invalid (department cannot be NULL)
```

## Output:

diff

employee_id	name	age	department
1	Alice	30	HR

## Explanation:

- **NOT NULL** (`department VARCHAR(50) NOT NULL`): Ensures that the `department` column always contains a value.
- **Why is it needed?:** It guarantees that every record has the necessary information, preventing incomplete data entries.

## 7. DEFAULT Constraint

The **DEFAULT** constraint sets a default value for a column if no value is explicitly provided during insertion.

### Example:

```
sql

-- Create a table with a DEFAULT constraint
CREATE TABLE products (
  product_id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  price DECIMAL(10, 2) NOT NULL,
  stock INT DEFAULT 10 -- Default stock is 10 if not specified
);

-- Insert data into the table without specifying stock
INSERT INTO products (name, price) VALUES ('New Product', 50.00);

-- Insert data into the table with all values specified
INSERT INTO products (name, price, stock) VALUES ('Another Product', 75.00, 20);
```

### Output:

```
mathematica

product_id | name           | price | stock
-----|-----|-----|-----
1          | New Product    | 50.00 | 10
2          | Another Product | 75.00 | 20
```

### Explanation:

- **DEFAULT** (`stock INT DEFAULT 10`): If no `stock` value is provided during insertion, it defaults to 10.
- **Why is it needed?:** It simplifies data entry by providing a sensible default for optional fields.