**PROMPT:** give me the detailed documentation of update command with examples in PostgreSQL

## **What is the UPDATE Command in PostgreSQL?**

The UPDATE command is used to **modify existing rows** in a table. You can update one or more columns in one or more rows based on a condition.

## Basic Syntax

UPDATE table\_name

SET column1 = value1,

column2 = value2,

...

WHERE condition;

### **Parameters**

* **table\_name**: Name of the table to update.
* **SET**: Specifies the columns and the new values.
* **WHERE**: Optional condition to specify which rows to update. If omitted, **all rows** in the table will be updated (risky).

**Table**

Let's use a sample table named employees:

CREATE TABLE employees (

emp\_id SERIAL PRIMARY KEY,

name VARCHAR(50),

department VARCHAR(50),

salary NUMERIC(10, 2)

);

INSERT INTO employees (name, department, salary) VALUES

('Alice', 'HR', 40000),

('Bob', 'IT', 50000),

('Charlie', 'IT', 55000),

('Diana', 'Finance', 45000);

**Examples of UPDATE in PostgreSQL**

### **1. Update a single row**

UPDATE employees

SET salary = 60000

WHERE name = 'Bob';

**Explanation**: Updates Bob's salary to 60000.

### **2. Update multiple columns**

UPDATE employees

SET salary = 47000, department = 'Accounts'

WHERE name = 'Diana';

**Explanation**: Updates Diana’s salary and department.

### **3. Update multiple rows**

UPDATE employees

SET salary = salary + 5000

WHERE department = 'IT';

**Explanation**: Increases salary by 5000 for all employees in the IT department.

### **4. Update all rows (without WHERE clause)**

UPDATE employees

SET department = 'General';

**Warning**: This changes the department column for **all rows** in the table. Use with caution.

### **5. Using a subquery in UPDATE**

UPDATE employees

SET salary = salary + 10000

WHERE emp\_id IN (

SELECT emp\_id FROM employees WHERE department = 'HR'

);

**Explanation**: Increases salary by 10000 for employees in the HR department.

### **6. Conditional Update with CASE**

UPDATE employees

SET salary = CASE

WHEN department = 'IT' THEN salary + 2000

WHEN department = 'Finance' THEN salary + 1000

ELSE salary

END;

**Explanation**: Applies different salary increments based on department.

## 

## **Best Practices**

* Always use a **WHERE clause** unless you intend to update all rows.
* Test with SELECT first to verify the rows to be affected:

SELECT \* FROM employees WHERE department = 'IT';

* Use RETURNING to view updated rows:

UPDATE employees

SET salary = salary + 1000

WHERE department = 'HR'

RETURNING \*;

## **Error Scenarios**

### **1. Forgetting WHERE clause**

UPDATE employees SET salary = 0;

-- All salaries will be set to 0!

### **2. Type mismatch**

UPDATE employees SET salary = 'high'; -- invalid input

## **PostgreSQL Features**

* UPDATE ... FROM — Useful when updating from another table (see below).
* UPDATE ... RETURNING — Returns the modified rows.

## **Example: UPDATE using JOIN (UPDATE ... FROM)**

Suppose we have a second table:

CREATE TABLE bonuses (

department VARCHAR(50),

bonus NUMERIC(10, 2)

);

INSERT INTO bonuses VALUES

('IT', 5000),

('HR', 3000);

Now, update the employees’ salaries based on bonus:

UPDATE employees

SET salary = salary + b.bonus

FROM bonuses b

WHERE employees.department = b.department;

**Explanation**: Joins both tables and applies the bonus to each matching employee.

## **Summary Table**

| **Feature** |  | **Example** |
| --- | --- | --- |
| Basic UPDATE |  | UPDATE table SET col = val WHERE cond; |
| Multiple Columns Update |  | SET col1 = val1, col2 = val2 |
| Subquery in WHERE |  | WHERE id IN (SELECT ...) |
| Subquery in SET |  | SET col = (SELECT ...) |
| Conditional Update (CASE) |  | SET col = CASE WHEN ... THEN ... END |
| UPDATE with JOIN (FROM) |  | UPDATE ... FROM ... WHERE ... |
| RETURNING Clause |  | UPDATE ... RETURNING \* |