Overview

- **Purpose**: Modifies existing data in one or more rows of a database table.
- **Syntax**: UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;
- **Key Concepts**:
    - Use WHERE clause to target specific rows; omit it to update all rows (use cautiously).
    - Supports updating single or multiple columns.
    - Conditions can use primary keys, other columns, comparisons (e.g., <, >), or combinations.
    - Arithmetic operations possible (e.g., age = age + 1).

Sample Table Setup

- **Database**: employees
- **Table**: People
    - Columns: person_id (SERIAL PRIMARY KEY), first_name (VARCHAR(50)), last_name (VARCHAR(50)), age (INT), city (VARCHAR(50))
- **Initial Data**: 20 rows with sample names, ages (24-45), and U.S. cities.

Initial Sample Output (abridged for brevity):

person_id	first_name	last_name	age	city
1	John	Doe	30	New York
2	Jane	Smith	25	Los Angeles
...	...	...	...	...
20	Stephanie	Scott	24	Orlando

Update Examples

1. Update Based on Primary Key

- **Query**: UPDATE People SET age = 40 WHERE person_id = 1;
- **Effect**: Targets exact row via unique ID.
- **Post-Update**: John's age changes to 40.

2. Update Based on Column Value (e.g., Last Name)

- **Query**: UPDATE People SET city = 'Austin' WHERE last_name = 'Smith';
- **Effect**: Updates all matching rows (here, Jane's city to Austin).
- **Note**: Case-sensitive; use ILIKE for case-insensitivity in some DBs.

3. Conditional Bulk Update (e.g., Age < 30)

- **Query**: UPDATE People SET age = 30 WHERE age < 30;
- **Effect**: Sets ages below 30 to exactly 30 (affects multiple rows: Jane, David, Jennifer, Daniel, Amanda, Stephanie).

4. Arithmetic Update (e.g., Increment Age in Specific City)

- **Query**: UPDATE People SET age = age + 1 WHERE city = 'New York';
- **Effect**: John's age becomes 41.
- **Tip**: Use for increments/decrements; supports expressions like age * 2.

5. Bulk Update Based on Condition (e.g., Age > 35)

- **Query**: UPDATE People SET first_name = 'Senior' WHERE age > 35;
- **Effect**: Changes first names to 'Senior' for older individuals (e.g., Michael, Robert, Maria, Christopher, Joseph).

6. Multi-Column Update

- **Query**: UPDATE People SET first_name = 'Ashish', last_name = 'Jangra', age = 25, city = 'India' WHERE person_id = 1;
- **Effect**: Overwrites multiple fields in one row (John → Ashish Jangra, 25, India).

Final Table State (After All Updates)

person_id	first_name	last_name	age	city
1	Ashish	Jangra	25	India
2	Jane	Smith	30	Austin
3	Senior	Johnson	40	Chicago
4	Emily	Brown	35	Houston
5	David	Jones	30	San Francisco
6	Sarah	Davis	32	Seattle
7	Senior	Wilson	45	Boston
8	Jennifer	Martinez	30	Miami
9	William	Taylor	38	Atlanta
10	Jessica	Anderson	33	Dallas
11	Daniel	Thomas	30	Philadelphia
12	Senior	Jackson	42	Phoenix
13	James	White	31	Denver

person_id	first_name	last_name	age	city
14	Elizabeth	Harris	36	Austin
15	Christopher	Clark	39	San Diego
16	Amanda	Lewis	30	Portland
17	Matthew	Walker	34	Detroit
18	Ashley	Allen	37	Las Vegas
19	Senior	Young	41	Nashville
20	Stephanie	Scott	30	Orlando

Best Practices

- **Always Test**: Use SELECT first to preview affected rows (e.g., SELECT * FROM People WHERE age < 30;).
- **Backup Data**: Run in transactions (BEGIN; UPDATE ...; ROLLBACK; to test).
- **Performance**: Index columns in WHERE for large tables.
- **Security**: Parameterize queries to avoid SQL injection.
- **DB Variations**: Syntax similar across MySQL/PostgreSQL/SQLite; check for DBMS-specific features (e.g., LIMIT in updates).

Conclusion

- UPDATE is essential for data maintenance.
- Combine with SELECT for verification.
- Practice on sample data to master conditions and bulk operations.