Data Maintenance with UPDATE

Mastering SQL with UPDATE Statement

Section 1: Learn

What is the UPDATE Statement?

The UPDATE statement in SQL is used to modify the existing values in one or more columns of one or more rows in a table. It is one of the most essential Data Manipulation Language (DML) commands used in real-world database management.

When and Why is UPDATE Used?

- To correct mistakes in stored data
- To refresh data based on real-world changes
- To alter values based on new business logic
- To apply calculations or migrate old values to new ones

How Does the UPDATE Statement Work?

- Specify the Table – Choose which table contains the data you want to update.
- Set New Values – Use the SET keyword to assign new values to specific columns.
- Use Conditions – Apply a WHERE clause to define exactly which row(s) should be updated.
- (Optional) Use Subqueries or Joins – Dynamically fetch and apply data from other tables.

Syntax Overview

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Important Notes

- WHERE Clause is Critical

  Without a WHERE clause, every row will be updated — often a catastrophic mistake!
- Multiple Columns Can Be Updated

  Comma-separate the column-value pairs inside the SET clause.
- Supports Expressions and Calculations

  You can apply operations like SET price = price * 0.9 to give discounts.
- Can Use Subqueries

  Update values based on values fetched from other tables.

Example Scenario:

```
UPDATE employees
SET department = 'Management', salary = 75000
WHERE employee_id = 1003;
```

Section 2: Practice

1. Update a Single Column

```
UPDATE books
SET price = 399
```

```
WHERE book_id = 22;
```

Sets the price of book with ID 22 to 399.

2. Update Multiple Columns

```
UPDATE students
SET grade = 'B+', attendance = 85
WHERE student_id = 1101;
```

Updates both grade and attendance for student 1101.

3. Update All Rows (No WHERE Clause!)

```
UPDATE products
SET stock = 0;
```

All products will now have stock = 0. Dangerous if unintended.

4. Conditional Update Using Logical Operators

```
UPDATE employees
SET bonus = 1000
WHERE department = 'Sales' AND experience > 3;
```

Grants a bonus to experienced employees in Sales.

5. Update with Calculated Values

```
UPDATE items
SET discount_price = original_price * 0.9
WHERE category = 'Electronics';
```

Applies a 10% discount on all electronic items.

6. Using Subquery in an UPDATE

```
UPDATE orders
SET shipping_city = (
  SELECT city FROM customers WHERE customers.customer_id =
orders.customer_id
)
WHERE order_date < '2023-01-01';
```

Copies city from customers table for older orders.

7. Update Using CASE (Conditional Logic)

```
UPDATE employees
SET bonus =
 CASE
   WHEN performance = 'Excellent' THEN 2000
   WHEN performance = 'Good' THEN 1000
   ELSE 500
 END;
```

Gives different bonuses based on performance.

Section 3: Know More

Frequently Asked Questions (FAQs)

1. **What happens if I forget the WHERE clause?**

   All rows will be updated. Always double-check before running UPDATE.

2. **Can I use a JOIN in UPDATE?**

   Yes! Many databases (like MySQL, PostgreSQL) support UPDATE with JOIN to update one table based on another.

```
UPDATE orders
JOIN customers ON orders.customer_id = customers.customer_id
SET orders.city = customers.city
WHERE orders.city IS NULL;
```

3. **Can I undo an UPDATE?**

   If you're using a database that supports transactions, you can ROLLBACK if you're in a transaction block. Otherwise, you'll need to manually reverse changes.

4. **Can I use functions or expressions in the SET clause?**

   Absolutely! You can use built-in functions like UPPER(), NOW(), ROUND() etc.

```
UPDATE users
SET username = UPPER(username);
```

5. **How can I test an UPDATE before running it?**

   Use a SELECT statement with the same WHERE clause to preview affected rows before updating.

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


UPDATE employees
SET department = 'Tech Support'
```

```
WHERE department = 'IT';
```