# Modifying Data in SQL

## Modifying Data in SQL: INSERT, DELETE, UPDATE

SQL provides three main commands for modifying data in tables:

---

## 1. INSERT

Used to add new rows into a table.

**Syntax:**

```sql
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

**Examples:**

- **Insert a single row:**

```sql
INSERT INTO employees (name, department_id, salary)
VALUES ('John Doe', 101, 50000);
```

- **Insert multiple rows:**

```sql
INSERT INTO employees (name, department_id, salary)
VALUES
    ('Alice Smith', 102, 60000),
    ('Bob Brown', 103, 70000);
```

- **Insert data into all columns:**

```sql
INSERT INTO employees
```

```sql
VALUES (1, 'Jane Doe', 101, 55000);
```

- **Insert data from another table:**

```sql
INSERT INTO employees_backup (employee_id, name, department_id, salary)
SELECT employee_id, name, department_id, salary
FROM employees
WHERE department_id = 101;
```

---

## 2. DELETE

Used to remove rows from a table.

**Syntax:**

```sql
DELETE FROM table_name
WHERE condition;
```

**Examples:**

- **Delete a specific row:**

```sql
DELETE FROM employees
WHERE employee_id = 1;
```

- **Delete rows based on a condition:**

```sql
DELETE FROM employees
WHERE salary < 30000;
```

- **Delete all rows (truncate-like operation):**

```sql
DELETE FROM employees;
```

**Note:** Use caution; this removes all rows from the table.

---

## 3. UPDATE

Used to modify existing rows in a table.

**Syntax:**

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

**Examples:**

- **Update a specific row:**

```sql
UPDATE employees
SET salary = 60000
WHERE employee_id = 1;
```

- **Update multiple rows:**

```sql
UPDATE employees
SET department_id = 104
WHERE department_id = 103;
```

- **Update all rows:**

```sql
UPDATE employees
```

```sql
SET salary = salary * 1.10; -- Increase all salaries by 10%
```

---

## Best Practices for Modifying Data

**1. Use Transactions for Safety**

Wrap `INSERT`, `DELETE`, or `UPDATE` statements in a transaction to ensure atomicity.

SQL

```sql
BEGIN TRANSACTION;

DELETE FROM employees
WHERE department_id = 105;

-- Rollback if something goes wrong
ROLLBACK;

-- Commit changes if everything is fine
COMMIT;
```

**2. Always Include a WHERE Clause for DELETE and UPDATE**

Avoid unintended changes or deletions.

- **Risky:**

SQL

```sql
DELETE FROM employees;
```

- **Safe:**

SQL

```sql
DELETE FROM employees
WHERE salary < 20000;
```

**3. Test with SELECT First**

Run a `SELECT` query with the same condition to ensure the correct rows are affected.

```sql
SELECT * FROM employees
WHERE department_id = 105;
```

**4. Use `RETURNING` (if supported by your database)**

Retrieve information about modified rows in databases like PostgreSQL.

```sql
UPDATE employees
SET salary = salary * 1.10
WHERE department_id = 102
RETURNING employee_id, salary;
```

---

## Combining Modifications

- **Insert and Update (`UPSERT` or MERGE):**
  - Insert a new row if it doesn't exist; otherwise, update it.
  - **PostgreSQL Example:**

```sql
INSERT INTO employees (employee_id, name, salary)
VALUES (1, 'John Doe', 60000)
ON CONFLICT (employee_id)
DO UPDATE SET salary = EXCLUDED.salary;
```

  - **MySQL Example:**

```sql
INSERT INTO employees (employee_id, name, salary)
VALUES (1, 'John Doe', 60000)
ON DUPLICATE KEY UPDATE salary = VALUES(salary);
```

- **Delete and Insert (Repopulate a Table):**

SQL

```sql
DELETE FROM employees_backup;
INSERT INTO employees_backup
SELECT * FROM employees;

```

---

Would you like specific examples tailored to your use case or additional queries for practicing?