

# DDL, DML & Constraints: CREATE, ALTER, INSERT, UPDATE, DELETE

Create and change tables with DDL (CREATE, ALTER, DROP, TRUNCATE), add, change and remove rows with DML (INSERT, UPDATE, DELETE, MERGE), protect data quality with constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK, DEFAULT, IDENTITY) and understand DELETE vs TRUNCATE vs DROP.

● SQL classes 2-3

✦ SQL Server 2022, 2026

Beginner

⌚ 120 min

## By the end of this lesson you can

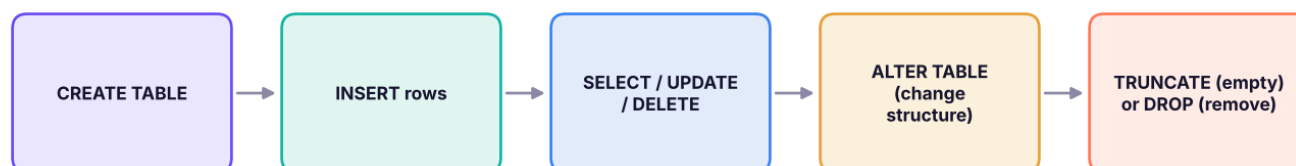
- Create databases and tables with appropriate data types and constraints
- Modify table structure with ALTER TABLE (add, change, drop columns and constraints)
- Insert single and multiple rows, insert from a query, update and delete rows safely
- Explain every constraint type and read the error each one raises
- Compare DELETE, TRUNCATE and DROP, and use MERGE for upserts

## Datasets used

`employees_small.csv`

## The life cycle of a table

### DDL and DML together



DDL creates and changes structure; DML fills and changes the rows inside it.

# CREATE TABLE

## SQL

```
CREATE TABLE departments (  
    dept_id    INT            PRIMARY KEY,  
    dept_name  VARCHAR(50)    NOT NULL UNIQUE,  
    location   VARCHAR(50)    DEFAULT 'Bengaluru'  
);  
  
CREATE TABLE staff (  
    emp_id     INT            PRIMARY KEY,  
    full_name  VARCHAR(100)   NOT NULL,  
    email      VARCHAR(100)   NOT NULL UNIQUE,  
    dept_id    INT            NOT NULL REFERENCES departments(dept_id),  
    salary     DECIMAL(10,2)  CHECK (salary > 0),  
    join_date  DATE           DEFAULT CURRENT_DATE,  
    is_active  BOOLEAN        DEFAULT TRUE  
);
```

## RESULT

Commands completed successfully.

Commands completed successfully.

In SQL Server you would usually write `IDENTITY(1,1)` for an auto-numbered key, `BIT` for the flag and `DEFAULT GETDATE()` for today. Here is the same `staff` table in full T-SQL style, with named constraints (names make error messages and later changes much easier):

## SQL

```
CREATE TABLE dbo.staff (  
    emp_id     INT IDENTITY(1,1) NOT NULL,  
    full_name  NVARCHAR(100)    NOT NULL,  
    email      VARCHAR(100)     NOT NULL,  
    dept_id    INT              NOT NULL,  
    salary     DECIMAL(10,2)    NULL,  
    join_date  DATE             NOT NULL CONSTRAINT df_staff_join  DEFAULT  
    (CAST(GETDATE() AS DATE)),  
    is_active  BIT              NOT NULL CONSTRAINT df_staff_active DEFAULT (1),  
    CONSTRAINT pk_staff        PRIMARY KEY (emp_id),  
    CONSTRAINT uq_staff_email  UNIQUE (email),  
    CONSTRAINT fk_staff_dept   FOREIGN KEY (dept_id) REFERENCES dbo.departments(dept_id),  
    CONSTRAINT ck_staff_salary CHECK (salary > 0)  
);
```

# Constraints at a glance

| Constraint               | Guarantees                                            | Example                                 |
|--------------------------|-------------------------------------------------------|-----------------------------------------|
| PRIMARY KEY              | Unique and not NULL; identifies each row              | emp_id                                  |
| FOREIGN KEY / REFERENCES | Value must exist in the parent table                  | dept_id → departments                   |
| UNIQUE                   | No duplicate values (NULL allowed once in SQL Server) | email                                   |
| NOT NULL                 | A value is required                                   | full_name                               |
| CHECK                    | Value must satisfy a condition                        | salary > 0 , discount BETWEEN 0 AND 0.9 |
| DEFAULT                  | Value used when none is supplied                      | is_active = 1                           |
| IDENTITY(seed, step)     | Auto-numbering (SQL Server)                           | IDENTITY(1,1)                           |

## INSERT: adding rows

SQL

```
INSERT INTO departments (dept_id, dept_name, location) VALUES
  (10, 'Finance', 'Mumbai'),
  (20, 'IT', 'Pune'),
  (30, 'HR', DEFAULT),
  (40, 'Sales', 'Delhi');

INSERT INTO staff (emp_id, full_name, email, dept_id, salary, join_date) VALUES
  (1, 'Kavya Iyer', 'kavya@abc.in', 30, 97000, '2016-02-28'),
  (2, 'Imran Reddy', 'imran@abc.in', 10, 157000, '2019-12-14'),
  (3, 'Harpreet Kulkarni', 'harpreet@abc.in', 10, 131000, '2024-10-05'),
  (4, 'Rahul Iyer', 'rahul@abc.in', 20, 112000, '2017-02-06'),
  (5, 'Isha Mehta', 'isha@abc.in', 40, 57000, '2021-09-07');

SELECT * FROM departments;
```

RESULT

(4 rows affected)

(5 rows affected)

| dept_id | dept_name | location  |
|---------|-----------|-----------|
| 10      | Finance   | Mumbai    |
| 20      | IT        | Pune      |
| 30      | HR        | Bengaluru |
| 40      | Sales     | Delhi     |

4 rows × 3 columns

#### SQL

```
SELECT emp_id, full_name, dept_id, salary, join_date, is_active FROM staff;
```

#### RESULT

| emp_id | full_name         | dept_id | salary  | join_date  | is_active |
|--------|-------------------|---------|---------|------------|-----------|
| 1      | Kavya Iyer        | 30      | 97,000  | 2016-02-28 | True      |
| 2      | Imran Reddy       | 10      | 157,000 | 2019-12-14 | True      |
| 3      | Harpreet Kulkarni | 10      | 131,000 | 2024-10-05 | True      |
| 4      | Rahul Iyer        | 20      | 112,000 | 2017-02-06 | True      |
| 5      | Isha Mehta        | 40      | 57,000  | 2021-09-07 | True      |

5 rows × 6 columns

DEFAULT filled HR's location with Bengaluru and every employee's `is_active` with TRUE.

#### **TIP** Always list the columns

`INSERT INTO staff VALUES (...)` without a column list depends on the column order and breaks the day someone adds a column. Always write `INSERT INTO staff (col1, col2, ...) VALUES (...)`.

## INSERT ... SELECT and SELECT ... INTO

Copy rows from a query into a table:

## SQL

```
-- Into an existing table
INSERT INTO staff_archive (emp_id, full_name, dept_id, salary)
SELECT emp_id, full_name, dept_id, salary
FROM staff
WHERE is_active = 0;

-- SQL Server: create a NEW table from a query (structure + data, no constraints)
SELECT emp_id, full_name, salary
INTO staff_backup_2026
FROM staff;
```

The portable equivalent of `SELECT ... INTO` is `CREATE TABLE new_table AS SELECT ...` (PostgreSQL, MySQL, Snowflake, DuckDB).

## What constraints do when rules are broken

Constraints reject bad data **at the database**, whatever app or person sends it. These statements would all fail in SQL Server with the messages shown (abridged):

| Statement                                                                     | Broken rule                     | SQL Server message                                                                                                     |
|-------------------------------------------------------------------------------|---------------------------------|------------------------------------------------------------------------------------------------------------------------|
| INSERT INTO departments<br>VALUES (10, 'Ops', 'Pune')                         | PRIMARY KEY                     | Violation of PRIMARY KEY constraint 'pk_departments'. Cannot insert duplicate key ... The duplicate key value is (10). |
| INSERT INTO staff (... , email,<br>...) VALUES (... , 'kavya@abc.in',<br>...) | UNIQUE                          | Violation of UNIQUE KEY constraint 'uq_staff_email'. Cannot insert duplicate key ...                                   |
| INSERT INTO staff (... ,<br>dept_id, ...) VALUES (... , 99, ...)              | FOREIGN KEY                     | The INSERT statement conflicted with the FOREIGN KEY constraint "fk_staff_dept".                                       |
| INSERT INTO staff (emp_id,<br>email, dept_id) VALUES<br>(9, 'x@abc.in', 10)   | NOT NULL                        | Cannot insert the value NULL into column 'full_name' ... column does not allow nulls. INSERT fails.                    |
| UPDATE staff SET salary =<br>-500 WHERE emp_id = 1                            | CHECK                           | The UPDATE statement conflicted with the CHECK constraint "ck_staff_salary".                                           |
| DELETE FROM departments WHERE<br>dept_id = 10                                 | FOREIGN KEY<br>(children exist) | The DELETE statement conflicted with the REFERENCE constraint "fk_staff_dept".                                         |

Our practice engine raises the same kind of error. For example, trying to add an employee to a department that does not exist:

### OUTPUT

Error: Constraint Error: Violates foreign key constraint because key "dept\_id: 99" does not exist in the referenced table

## UPDATE: changing rows

### SQL

```
-- Give everyone in Finance a 5% raise
UPDATE staff
SET    salary = salary * 1.05
WHERE  dept_id = 10;

-- Change two columns for one person
UPDATE staff
SET    dept_id = 20, salary = 60000
WHERE  emp_id = 5;

SELECT emp_id, full_name, dept_id, salary FROM staff ORDER BY emp_id;
```

### RESULT

(2 rows affected)

(1 row affected)

| emp_id | full_name         | dept_id | salary  |
|--------|-------------------|---------|---------|
| 1      | Kavya Iyer        | 30      | 97,000  |
| 2      | Imran Reddy       | 10      | 164,850 |
| 3      | Harpreet Kulkarni | 10      | 137,550 |
| 4      | Rahul Iyer        | 20      | 112,000 |
| 5      | Isha Mehta        | 20      | 60,000  |

5 rows × 4 columns

### **WATCH OUT** UPDATE or DELETE without WHERE changes EVERY row

UPDATE staff SET salary = 0; sets every salary to zero. Before any UPDATE or DELETE:

1. Write the WHERE first and run it as a SELECT to see exactly which rows will change.
2. Wrap the change in a transaction (BEGIN TRAN ... then check, then COMMIT or ROLLBACK) (Lesson 14).
3. On production data, take a backup or work in a copy.

## UPDATE with a join (SQL Server)

## SQL

```
-- Mark staff in Mumbai departments as 'on-site'
UPDATE s
SET     s.work_mode = 'on-site'
FROM    staff AS s
JOIN    departments AS d ON d.dept_id = s.dept_id
WHERE   d.location = 'Mumbai';
```

## DELETE: removing rows

### SQL

```
DELETE FROM staff
WHERE emp_id = 5;

SELECT COUNT(*) AS staff_left FROM staff;
```

### RESULT

(1 row affected)

| staff_left |
|------------|
|------------|

|   |
|---|
| 4 |
|---|

1 rows × 1 columns

## ALTER TABLE: changing structure

### SQL

```
ALTER TABLE staff ADD COLUMN phone VARCHAR(15);
ALTER TABLE staff ADD COLUMN work_mode VARCHAR(10) DEFAULT 'hybrid';
SELECT emp_id, full_name, phone, work_mode FROM staff;
```

### RESULT

| emp_id | full_name         | phone | work_mode |
|--------|-------------------|-------|-----------|
| 1      | Kavya Iyer        | NULL  | hybrid    |
| 2      | Imran Reddy       | NULL  | hybrid    |
| 3      | Harpreet Kulkarni | NULL  | hybrid    |
| 4      | Rahul Iyer        | NULL  | hybrid    |

4 rows × 4 columns

The SQL Server forms of the common changes:

#### SQL

```

ALTER TABLE staff ADD phone VARCHAR(15) NULL;           -- add a column (no
COLUMN keyword in T-SQL)
ALTER TABLE staff ALTER COLUMN full_name NVARCHAR(150) NOT NULL; -- change type /
nullability
ALTER TABLE staff DROP COLUMN phone;                     -- remove a column
ALTER TABLE staff ADD CONSTRAINT ck_staff_mode CHECK (work_mode IN ('remote','hybrid','on-
site'));
ALTER TABLE staff DROP CONSTRAINT ck_staff_mode;         -- remove a constraint
by name
EXEC sp_rename 'staff.full_name', 'employee_name', 'COLUMN'; -- rename a column
EXEC sp_rename 'staff', 'employees_master';               -- rename a table

```

#### **NOTE** Changing a column that has data

Making a column smaller (VARCHAR(100) → VARCHAR(10)) fails if existing values do not fit; adding NOT NULL fails if NULLs exist. Clean the data first, or add a DEFAULT.

## DELETE vs TRUNCATE vs DROP

This is one of the most common SQL interview questions.

|                                     | DELETE                            | TRUNCATE TABLE                                     | DROP TABLE                                                                  |
|-------------------------------------|-----------------------------------|----------------------------------------------------|-----------------------------------------------------------------------------|
| Family                              | DML                               | DDL                                                | DDL                                                                         |
| Removes                             | Some or all <b>rows</b>           | <b>All rows</b>                                    | The <b>whole table</b> : rows, structure, indexes, constraints, permissions |
| WHERE allowed                       | Yes                               | No                                                 | No                                                                          |
| Speed on large tables               | Slower: logs each row             | Very fast: deallocates pages                       | Fast                                                                        |
| Logging                             | Every row logged                  | Minimal (page deallocations)                       | Minimal                                                                     |
| Fires DELETE triggers               | Yes                               | No                                                 | No                                                                          |
| Resets IDENTITY                     | No                                | <b>Yes</b> , back to the seed                      | Table is gone                                                               |
| Works when referenced by a FK       | Yes (if no child rows violate it) | <b>No</b>                                          | <b>No</b> (drop the FK first)                                               |
| Can be rolled back in a transaction | Yes                               | Yes in SQL Server (inside an explicit transaction) | Yes in SQL Server (inside an explicit transaction)                          |

## SQL

```
CREATE TABLE temp_log (id INT, msg VARCHAR(50));
INSERT INTO temp_log VALUES (1, 'start'), (2, 'load'), (3, 'end');
DELETE FROM temp_log WHERE id = 2;           -- removes one row
SELECT COUNT(*) AS after_delete FROM temp_log;
```

## RESULT

Commands completed successfully.  
 (3 rows affected)  
 (1 row affected)

**after\_delete**

2

1 rows × 1 columns

## SQL

```
TRUNCATE TABLE temp_log;                                -- removes all rows, keeps the table
SELECT COUNT(*) AS after_truncate FROM temp_log;
```

## RESULT

| after_truncate |
|----------------|
|----------------|

|   |
|---|
| 0 |
|---|

1 rows × 1 columns

## SQL

```
DROP TABLE temp_log;                                    -- removes the table itself
SELECT COUNT(*) AS tables_named_temp_log FROM information_schema.tables WHERE table_name =
'temp_log';
```

## RESULT

| tables_named_temp_log |
|-----------------------|
|-----------------------|

|   |
|---|
| 0 |
|---|

1 rows × 1 columns

In SQL Server, `DROP TABLE IF EXISTS temp_log;` avoids an error when the table is already gone.

## MERGE: insert or update in one statement (upsert)

Data pipelines often receive a file of changes: some rows are new, others update existing rows.

MERGE handles both:

## SQL

```
MERGE dbo.staff AS target
USING dbo.staff_updates AS src
    ON target.emp_id = src.emp_id
WHEN MATCHED THEN
    UPDATE SET target.salary = src.salary, target.dept_id = src.dept_id
WHEN NOT MATCHED BY TARGET THEN
    INSERT (emp_id, full_name, email, dept_id, salary)
    VALUES (src.emp_id, src.full_name, src.email, src.dept_id, src.salary)
WHEN NOT MATCHED BY SOURCE THEN
    UPDATE SET target.is_active = 0;                        -- in staff but not in the file
```

# Loading CSV files with SQL

## SQL

```
CREATE TABLE dbo.employees_small (  
    emp_id VARCHAR(10) PRIMARY KEY, name NVARCHAR(100), department VARCHAR(30),  
    city VARCHAR(30), join_date DATE, salary DECIMAL(10,2), rating TINYINT  
);  
  
BULK INSERT dbo.employees_small  
FROM 'C:\Data\employees_small.csv'  
WITH (FORMAT = 'CSV', FIRSTROW = 2, FIELDTERMINATOR = ',', ROWTERMINATOR = '0x0a');
```

The SSMS **Import Flat File** wizard is easier for one-off loads; BULK INSERT, bcp or an ETL tool (SSIS, Azure Data Factory, Fabric pipelines) are used for repeatable loads.

### DO IT WITH AI Do it with AI

Ask: "Write SQL Server CREATE TABLE statements with constraints for a hospital database: patients, doctors, appointments (a patient can see many doctors; a doctor sees many patients). Include primary keys, foreign keys, CHECK constraints for appointment status, and sensible data types." Then review it: are the keys right, is money DECIMAL, are dates DATE/DATETIME2, are the constraints named?

## Summary

### KEY TAKEAWAYS

- **DDL**: CREATE, ALTER, DROP, TRUNCATE change structure. **DML**: INSERT, UPDATE, DELETE, MERGE change rows.
- Constraints (PK, FK, UNIQUE, NOT NULL, CHECK, DEFAULT, IDENTITY) protect data quality at the source; name them.
- Always list columns in INSERT; always test the WHERE of an UPDATE/DELETE with a SELECT first, inside a transaction.
- DELETE removes chosen rows (logged, triggers fire); TRUNCATE empties the table fast and resets IDENTITY; DROP removes the table.
- MERGE performs upserts; BULK INSERT and the Import wizard load files.

### Quick quiz

6 questions · tap an answer to check it

**1. Which statement removes all rows but keeps the table structure and resets the IDENTITY counter?**

**A** DELETE FROM t

**B** TRUNCATE TABLE t

**C** DROP TABLE t

**D** ALTER TABLE t

**Answer: B.** TRUNCATE empties the table quickly and resets IDENTITY to its seed.

**2. You run UPDATE staff SET salary = 50000; by mistake. What happened?**

**A** Only the first row changed

**B** Every row's salary is now 50,000

**C** Nothing, because there is no WHERE

**D** An error is raised

**Answer: B.** Without WHERE, UPDATE changes every row. Use transactions and test with SELECT first.

**3. Which constraint stops a salary of -500?**

**A** UNIQUE

**B** DEFAULT

**C** CHECK (salary > 0)

**D** FOREIGN KEY

**Answer: C.** CHECK validates values against a condition.

**4. Inserting an employee with dept\_id 99 fails with a FOREIGN KEY error. Why?**

- A** dept\_id must be unique
- B** No department with id 99 exists in the parent table
- C** dept\_id cannot be a number
- D** The employee table is full

**Answer: B.** A foreign key value must exist in the referenced table.

**5. Which is TRUE about DELETE?**

- A** It can remove selected rows with WHERE and fires DELETE triggers
- B** It is DDL
- C** It resets IDENTITY
- D** It removes the table structure

**Answer: A.** DELETE is DML, works row by row with a WHERE, and fires triggers.

**6. How do you add a column phone to staff in SQL Server?**

- A** ALTER TABLE staff ADD COLUMN phone VARCHAR(15) only
- B** ALTER TABLE staff ADD phone VARCHAR(15)
- C** UPDATE staff ADD phone
- D** INSERT COLUMN phone INTO staff

**Answer: B.** T-SQL uses ADD without the COLUMN keyword (PostgreSQL/MySQL accept ADD COLUMN).

### EXERCISE Build and populate a small schema

Create projects(project\_id PK, name UNIQUE NOT NULL, budget DECIMAL(12,2) CHECK budget ≥ 0, start\_date DATE, status with CHECK IN ('planned','active','closed') DEFAULT 'planned') and assignments(project\_id FK, emp\_id FK, role, hours, PRIMARY KEY (project\_id, emp\_id)). Insert 3 projects and 4 assignments, raise the budget of active projects by 10%, close one project, and delete assignments of closed projects.

### Show solution (worked answer)

#### SQL

```
CREATE TABLE projects (  
  project_id INT PRIMARY KEY, name VARCHAR(80) NOT NULL UNIQUE,  
  budget DECIMAL(12,2) CHECK (budget ≥ 0), start_date DATE,  
  status VARCHAR(10) DEFAULT 'planned' CHECK (status IN  
  ('planned','active','closed')));  
CREATE TABLE assignments (  
  project_id INT REFERENCES projects(project_id), emp_id INT REFERENCES staff(emp_id),  
  role VARCHAR(30), hours INT CHECK (hours > 0), PRIMARY KEY (project_id, emp_id));  
INSERT INTO projects (project_id, name, budget, start_date, status) VALUES  
  (1,'CRM revamp',2500000,'2026-01-10','active'), (2,'Payroll audit',600000,'2026-03-  
01','active'), (3,'Data lake',4000000,'2026-06-01','planned');  
INSERT INTO assignments VALUES (1,1,'Analyst',120),(1,4,'Developer',300),  
(2,2,'Lead',80),(3,3,'Architect',40);  
UPDATE projects SET budget = budget * 1.10 WHERE status = 'active';  
UPDATE projects SET status = 'closed' WHERE project_id = 2;  
DELETE FROM assignments WHERE project_id IN (SELECT project_id FROM projects WHERE  
status = 'closed');
```

## INTERVIEW CORNER

### Questions you may be asked

**Q: DELETE vs TRUNCATE vs DROP?** DELETE (DML) removes chosen rows with WHERE, logs each row, fires triggers and keeps IDENTITY. TRUNCATE (DDL) removes all rows quickly, minimal logging, resets IDENTITY, no WHERE, not allowed when referenced by a FK. DROP removes the whole table and its definition.

**Q: What are constraints and why use them instead of app validation?** Rules enforced by the database (PK, FK, UNIQUE, NOT NULL, CHECK, DEFAULT). They protect data no matter which app, script or person writes to the table.

**Q: How do you safely run an UPDATE on production?** Run the WHERE as a SELECT first, wrap in BEGIN TRAN, check @@ROWCOUNT and the result, then COMMIT or ROLLBACK; have a backup.

## Practise more online

Free, trusted places to test yourself on this topic. Links open the original site in a new tab.