UNIT 2 – Structured Query Language (SQL)

2.1 Introduction to SQL

Subtopics

- Importance of SQL
- SQL Categories
- Basic Syntax Rules

Learning Outcome

By the end of this topic, you will be able to:

- Understand what SQL is and why it is the standard language for databases.
- Identify the different types of SQL commands.
- Recognize SQL syntax conventions for writing and executing queries.

Concept Explanation

Databases store data, but without a language to communicate with them, they are just containers. **Structured Query Language (SQL)** is the universal language used to manage and manipulate data stored in relational databases.

SQL allows you to **create**, **modify**, **query**, and **control** data — it acts as a bridge between users and the DBMS.

What is SQL?

Definition:
Structured Query Language (SQL) is a **standardized language** used to interact with relational databases through operations such as creation, insertion, querying, updating, and deletion of data.

Developed by: IBM in the 1970s (originally called SEQUEL).
Standardized by: ANSI (American National Standards Institute) and ISO.

Why SQL is Important

1. **Universal Language:** Works across all RDBMS (MySQL, Oracle, PostgreSQL, SQL Server).
2. **Powerful and Simple:** Allows complex data operations in simple commands.
3. **Data Control:** Offers access control, integrity enforcement, and transaction management.
4. **Integration:** Essential in full-stack development—every backend stack communicates with SQL databases.
5. **Industry Standard:** Required in data analysis, backend programming, and system design.

SQL Categories

SQL commands are grouped into five main categories based on their function.

Category	Full Form	Purpose	Example Commands
DDL	Data Definition Language	Defines database structure	CREATE, ALTER, DROP, TRUNCATE
DML	Data Manipulation Language	Manages data inside tables	INSERT, UPDATE, DELETE
DQL	Data Query Language	Retrieves data from tables	SELECT
DCL	Data Control Language	Controls access rights	GRANT, REVOKE
TCL	Transaction Control Language	Manages transactions	COMMIT, ROLLBACK, SAVEPOINT

Basic SQL Syntax Rules

1. **Case Insensitive:**
   SQL keywords are not case sensitive (SELECT = select), but uppercase is preferred for readability.
2. **Statements End with Semicolon (;):**
   SELECT * FROM students;
3. **String Literals Use Single Quotes:**
   INSERT INTO students VALUES (101, 'Arjun', 'CSE');
4. **Identifiers (table and column names):**
   - Should not contain spaces.
   - Usually lowercase or snake_case for consistency.
     Example: student_name, course_id.
7. **Comments:**
   -- This is a single-line comment
   /* This is a
   multi-line comment */

Example Interaction with Database

```
CREATE DATABASE college;
USE college;

CREATE TABLE student (
    roll_no INT PRIMARY KEY,
    name VARCHAR(50),
    department VARCHAR(30)
);

INSERT INTO student VALUES (101, 'Kavya', 'CSE');
SELECT * FROM student;
```

Output:

roll_no	name	department
101	Kavya	CSE

This example demonstrates SQL's core capabilities: defining structure (DDL), inserting data (DML), and querying data (DQL).

Best Practices

- Use **uppercase** for SQL keywords and **lowercase** for identifiers.
- Indent and format queries clearly.
- Test queries with limited records before large updates or deletions.
- Use comments to document logic in complex scripts.

Checkpoint Review

1. Define SQL in one sentence.
2. Name the five categories of SQL commands.
3. Which category does SELECT belong to?
4. What does TCL manage in SQL?
5. Write a simple SQL command to create a table named employee with columns (id, name, salary).

Reflection Prompt

Think of a small real-world system (library, store, or college).
What tables might you need, and what SQL operations would you perform on them daily?

Summary

- SQL is the standard language for interacting with relational databases.
- It is divided into categories—DDL, DML, DQL, DCL, and TCL.
- SQL syntax is straightforward and uniform across most database systems.
- Mastering SQL is essential for developers, data analysts, and administrators.

2.2 Data Definition Language (DDL)

Subtopics

- CREATE
- ALTER
- DROP
- TRUNCATE
- Table Design and Structure

Learning Outcome

By the end of this topic, you should be able to:

- Understand how to define and modify database objects using DDL commands.
- Create, alter, and delete tables and databases.
- Identify the difference between DROP and TRUNCATE.

Concept Explanation

The **Data Definition Language (DDL)** in SQL is used to define the **structure** of the database—its tables, schemas, and relationships.
DDL commands create, modify, or remove database objects such as databases, tables, indexes, and views.

Unlike data manipulation commands (DML), DDL commands **change the schema** and are automatically committed (cannot be rolled back in most RDBMS).

Major DDL Commands

Command	Purpose	Example
CREATE	Create new database or table.	CREATE TABLE student (...);
ALTER	Modify existing table structure.	ALTER TABLE student ADD column;
DROP	Delete a table or database permanently.	DROP TABLE student;
TRUNCATE	Remove all data from a table but keep its structure.	TRUNCATE TABLE student;

1. CREATE Command

The CREATE statement is used to define a new database or a table.

Create Database

```
CREATE DATABASE college;
```

Use Database

```
USE college;
```

Create Table

```
CREATE TABLE student (
    roll_no INT PRIMARY KEY,
    name VARCHAR(50),
    department VARCHAR(30),
    email VARCHAR(50),
    mark INT
);
```

Explanation:

- INT, VARCHAR define data types.
- PRIMARY KEY ensures unique identification.

Output:

Query OK, 0 rows affected

2. ALTER Command

Used to **modify** an existing table's structure without deleting data.

Add a Column

```
ALTER TABLE student ADD phone VARCHAR(15);
```

Modify Data Type of a Column

```
ALTER TABLE student MODIFY mark DECIMAL(5,2);
```

Rename a Column

```
ALTER TABLE student RENAME COLUMN department TO dept;
```

Drop a Column

```
ALTER TABLE student DROP COLUMN email;
```

3. DROP Command

The DROP command **deletes an entire table or database** permanently along with all its data and structure.

Drop Table

```
DROP TABLE student;
```

Drop Database

```
DROP DATABASE college;
```

Caution:
Once dropped, recovery requires restoring from backup—DROP cannot be rolled back.

4. TRUNCATE Command

The TRUNCATE command **removes all records** from a table but **retains the table structure** for reuse.
It is faster than DELETE because it doesn't log individual row deletions.

Example

```
TRUNCATE TABLE student;
```

Effect:

- All data deleted instantly.
- Table structure remains for new inserts.

Difference from DROP:

TRUNCATE	DROP
Deletes only data, keeps structure.	Deletes data and structure.
Faster, minimal logging.	Complete removal.
Table remains accessible.	Table no longer exists.

Table Design Guidelines

When designing tables, follow good practices:

1. Choose appropriate data types (use smallest required size).
2. Define primary keys for unique identification.
3. Use meaningful table and column names.
4. Apply constraints where necessary (e.g., NOT NULL, UNIQUE).
5. Avoid redundant or repeating columns.

Example:

```
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50) NOT NULL,
    dept VARCHAR(30),
    salary DECIMAL(10,2) DEFAULT 0
);
```

Hands-On Practice (MySQL)

- **Create a Database and Table**

```
CREATE DATABASE company_db;
USE company_db;

CREATE TABLE department (
    dept_id CHAR(3) PRIMARY KEY,
    dept_name VARCHAR(40) NOT NULL
);
```

- **Alter Table**

```
ALTER TABLE department ADD location VARCHAR(40);
```

- **Truncate and Drop**

```
TRUNCATE TABLE department;
DROP TABLE department;
DROP DATABASE company_db;
```

Checkpoint Review

1. What is the difference between DDL and DML commands?
2. Write SQL statements to create a database named training.
3. Which command will you use to remove a table but keep its structure?
4. How does ALTER differ from CREATE?
5. List two table design best practices.

Reflection Prompt

Suppose your company expands and you need to add a branch_location column to the existing employee table.
Which command would you use, and how would you verify the modification?

Summary

- **DDL** defines and modifies the structure of databases and tables.
- Major commands: CREATE, ALTER, DROP, TRUNCATE.
- DDL changes are automatically committed.
- Careful table design ensures scalability and data integrity.
- Proper use of DDL simplifies database maintenance and evolution.

2.3 Data Manipulation Language (DML)

Subtopics

- INSERT
- UPDATE
- DELETE
- WHERE Clause

Learning Outcome

By the end of this topic, you will be able to:

- Perform data manipulation operations in MySQL using DML commands.
- Add, update, and delete records in tables.
- Use the WHERE clause to filter and modify specific rows safely.

Concept Explanation

The **Data Manipulation Language (DML)** includes SQL commands used to manage data stored within tables.
While DDL defines *structure*, DML focuses on *content*.

DML operations are used daily by developers and database administrators to modify information without altering the schema.

Main DML Commands

Command	Purpose	Example
INSERT	Adds new data into a table.	INSERT INTO student VALUES (...);
UPDATE	Modifies existing records.	UPDATE student SET mark=90;
DELETE	Removes one or more records.	DELETE FROM student WHERE roll_no=101;

These commands directly affect data and can be rolled back using transaction control (ROLLBACK) if not committed.

1. INSERT Command

The INSERT command adds new rows (records) to a table.

Syntax

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

Example

```
CREATE DATABASE dml_demo;
USE dml_demo;

CREATE TABLE student (
    roll_no INT PRIMARY KEY,
    name VARCHAR(50),
    department VARCHAR(30),
    mark INT
```

```
);

INSERT INTO student (roll_no, name, department, mark)
VALUES (101, 'Arjun', 'CSE', 88);
```

Output:

```
Query OK, 1 row affected
```

Insert Multiple Rows

```
INSERT INTO student VALUES
(102, 'Kavya', 'IT', 92),
(103, 'Meera', 'ECE', 81);
```

Best Practices

- Always list column names in INSERT for clarity.
- Match the number and type of values to columns.

2. UPDATE Command

The UPDATE command modifies existing records in a table.

Syntax

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

Example

```
UPDATE student
SET mark = 95
WHERE roll_no = 101;
```

Without WHERE:

```
UPDATE student SET mark = 100;
```

Caution: Omitting WHERE updates **all rows** in the table.

Check Update:

```
SELECT * FROM student WHERE roll_no = 101;
```

Best Practices

- Always verify your WHERE condition before executing.
- Use SELECT first to confirm which rows will be affected.

3. DELETE Command

The DELETE command removes one or more rows from a table.

Syntax

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

Example

```
DELETE FROM student
WHERE roll_no = 103;
```

Delete All Rows (Use Carefully):

```
DELETE FROM student;
```

Difference between DELETE and TRUNCATE:

DELETE	TRUNCATE
Deletes row by row.	Deletes all rows at once.
Can use WHERE clause.	Cannot use WHERE.
Slower; logs each deletion.	Faster; minimal logging.
Can be rolled back.	Cannot be rolled back (DDL).

4. WHERE Clause

The WHERE clause filters records that meet specific conditions. It is used with SELECT, UPDATE, and DELETE.

Syntax

```
SELECT column1, column2 FROM table_name
WHERE condition;
```

Common Conditions

Operator	Description	Example
=	Equal to	`WHERE dept = 'CSE'`
<> or !=	Not equal to	`WHERE mark <> 90`

> < >= <=	Comparison	WHERE mark >= 80
BETWEEN	Range	WHERE mark BETWEEN 80 AND 90
IN	Matches any value in list	WHERE dept IN ('CSE','IT')
LIKE	Pattern match	WHERE name LIKE 'A%'

Example

```
SELECT name, mark
FROM student
WHERE mark >= 85;
```

Hands-On Practice (MySQL)

1. **Create and Populate Table**

```
CREATE DATABASE employee_db;
USE employee_db;

CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    department VARCHAR(30),
    salary DECIMAL(10,2)
);

INSERT INTO employee VALUES
(1, 'Meera', 'HR', 55000),
(2, 'Arjun', 'IT', 70000),
(3, 'Kavya', 'Finance', 60000);
```

2. **Update Salary**

```
UPDATE employee SET salary = salary + 5000
WHERE department = 'IT';
```

3. **Delete Record**

```
DELETE FROM employee
WHERE emp_id = 3;
```

4. **View Result**

```
SELECT * FROM employee;
```

Checkpoint Review

1. Write a query to insert a new student (104, 'Ravi', 'CSE', 89).
2. How can you increase all marks by 5 using SQL?
3. What happens if you run an UPDATE without a WHERE clause?
4. Write a query to delete all students from the ECE department.
5. What is the difference between DELETE and TRUNCATE?

Reflection Prompt

If you are managing a “Library” database, how would you handle issuing and returning books using INSERT, UPDATE, and DELETE?

Summary

- **DML** manages data within tables.
- INSERT adds records; UPDATE modifies them; DELETE removes them.
- The WHERE clause controls which rows are affected.
- Always test queries with SELECT first to avoid unintended data loss.
- DML operations can be rolled back before commit, ensuring data safety.

2.4 Data Query Language (DQL)

Subtopics

- SELECT Statement
- WHERE Clause
- ORDER BY, DISTINCT, LIMIT
- BETWEEN, IN, LIKE
- Column Aliases

Learning Outcome

After completing this topic, you should be able to:

- Retrieve and filter data from tables using SELECT.
- Sort, limit, and refine query results.
- Use operators and patterns to extract meaningful information efficiently.

Concept Explanation

The **Data Query Language (DQL)** is used to query and retrieve data from the database.
In SQL, **SELECT** is the primary command for extracting specific information according to given conditions.

DQL focuses on **"reading"** data, not modifying it — making it one of the safest and most frequently used SQL categories.

1. SELECT Statement

The SELECT command is used to retrieve data from one or more tables.

Syntax

```
SELECT column1, column2, ...
FROM table_name;
```

Example

```
SELECT name, department
FROM student;
```

Output:

name	department
Arjun	CSE
Kavya	IT
Meera	ECE

Select All Columns

```
SELECT * FROM student;
```

2. WHERE Clause

The WHERE clause filters records based on conditions.

Syntax

```
SELECT column1, column2
FROM table_name
WHERE condition;
```

Example

```
SELECT name, mark
FROM student
WHERE department = 'CSE';
```

Common Comparison Operators

Operator	Meaning	Example
=	Equal to	WHERE mark = 90
<> or !=	Not equal to	WHERE dept != 'IT'
> < >= <=	Comparison	WHERE mark > 80
BETWEEN	Range	WHERE mark BETWEEN 75 AND 90
IN	Matches multiple values	WHERE dept IN ('CSE','IT')
LIKE	Pattern matching	WHERE name LIKE 'A%'
IS NULL	Tests for null values	WHERE email IS NULL

3. ORDER BY Clause

Used to sort query results either in ascending (default) or descending order.

Syntax

```
SELECT column1, column2
FROM table_name
ORDER BY column1 [ASC|DESC];
```

Example

```
SELECT name, mark
FROM student
ORDER BY mark DESC;
```

Output:

Records are displayed in decreasing order of marks.

4. DISTINCT Keyword

Used to eliminate duplicate values from the result set.

Syntax

```
SELECT DISTINCT column_name
FROM table_name;
```

Example

```
SELECT DISTINCT department
FROM student;
```

Output:

```
CSE
IT
ECE
```

Only unique departments are displayed.

5. LIMIT Clause

Used to restrict the number of rows returned by a query (useful for pagination or sampling).

Syntax

```
SELECT column1, column2
FROM table_name
LIMIT n;
```

Example

```
SELECT * FROM student LIMIT 3;
Output:
Displays only the first 3 records.
```

6. BETWEEN Operator

Retrieves data between a specific range of values.

Syntax

```
SELECT column_name
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example

```
SELECT name, mark
FROM student
WHERE mark BETWEEN 80 AND 90;
```

7. IN Operator

Used when a column matches any value in a given list.

Syntax

```
SELECT column_name
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Example

```
SELECT name, department
FROM student
WHERE department IN ('CSE', 'IT');
```

8. LIKE Operator

Used for pattern matching in text columns.

Wildcards

Symbol	Meaning	Example
%	Represents any number of characters	'A%' → names starting with A
_	Represents a single character	'A_' → two-letter names starting with A

Example

```
SELECT name
FROM student
WHERE name LIKE 'A%';
Output:
Displays all names starting with "A".
```

9. Aliases

Aliases provide temporary names for tables or columns — helpful for readability and clarity.

Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

Example

```
SELECT name AS StudentName, department AS Dept
FROM student;
```

Output:

StudentName	Dept

Arjun	CSE
Kavya	IT

Hands-On Practice (MySQL)

1. **Create and Insert Data**

```
CREATE DATABASE dql_demo;
USE dql_demo;

CREATE TABLE student (
    roll_no INT PRIMARY KEY,
    name VARCHAR(50),
    department VARCHAR(30),
    mark INT
);

INSERT INTO student VALUES
(101, 'Arjun', 'CSE', 89),
(102, 'Kavya', 'IT', 95),
(103, 'Meera', 'ECE', 84),
(104, 'Rahul', 'CSE', 91),
(105, 'Divya', 'IT', 89);
```

2. **Retrieve and Sort**

```
SELECT name, department, mark
FROM student
WHERE mark >= 85
ORDER BY mark DESC;
```

3. **Distinct and Limit**

```
SELECT DISTINCT department FROM student;
SELECT * FROM student LIMIT 3;
```

4. **Pattern Match**

```
SELECT name FROM student WHERE name LIKE 'D%';
```

Checkpoint Review

1. What is the function of the SELECT statement?
2. How does the DISTINCT keyword work?
3. Write a query to display all students from CSE or IT departments.
4. How do you sort student names in descending order?
5. What will SELECT * FROM student WHERE mark BETWEEN 80 AND 90; return?

Reflection Prompt

Consider an “Employee” table with salary details.
How would you query to find the top three highest-paid employees?
Which clauses would you use?

Summary

- **DQL** retrieves data from databases using SELECT.
- The WHERE clause filters results; ORDER BY sorts them.
- DISTINCT, LIMIT, BETWEEN, IN, and LIKE refine output.
- Aliases improve readability for columns and tables.
- DQL is the foundation of data analysis, reporting, and application queries.

2.5 Constraints (Enhanced)

Subtopics

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- PRIMARY KEY
- FOREIGN KEY
- AUTO_INCREMENT
- ENUM
- INDEX
- ON DELETE / ON UPDATE (Referential Actions)
- Adding and Removing Constraints

Learning Outcome

After completing this topic, you will be able to:

- Understand all SQL constraints and their use cases.
- Enforce data integrity and consistency through rules at the schema level.
- Apply advanced constraint options such as cascading and enumerations.

Concept Explanation

A **constraint** defines a rule that restricts what data can be stored in a column.
Constraints maintain **data integrity**, ensure **validity**, and define **relationships** between tables.
Without constraints, data could become inconsistent, redundant, or logically invalid.

Core Constraints Overview

Constraint	Purpose	Scope
NOT NULL	Ensures no null values are stored.	Column level
UNIQUE	Ensures all values are unique.	Column / Table
DEFAULT	Assigns default value when none given.	Column
CHECK	Ensures data meets logical conditions.	Column / Table
PRIMARY KEY	Uniquely identifies each record.	Table
FOREIGN KEY	Creates and enforces relationships.	Table
AUTO_INCREMENT	Automatically generates sequential IDs.	Column
ENUM	Restricts values to a predefined set.	Column
INDEX	Improves query performance (not integrity).	Column
ON DELETE / UPDATE	Defines cascading referential actions.	Relationship

1. NOT NULL Constraint

Ensures that a column cannot contain NULL values.

```
CREATE TABLE student (
    roll_no INT NOT NULL,
    name VARCHAR(50) NOT NULL
);
```

If a NULL value is inserted into name, MySQL throws an error.

2. UNIQUE Constraint

Guarantees that all values in a column are unique across rows.

```
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    email VARCHAR(50) UNIQUE
);
```

No two employees can share the same email address.

3. DEFAULT Constraint

Assigns a predefined value if none is supplied during insertion.

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    status VARCHAR(20) DEFAULT 'Pending'
);
```

If no status is given, 'Pending' is stored automatically.

4. CHECK Constraint

Ensures that column values satisfy a logical condition.

```
CREATE TABLE marks (
    student_id INT,
    mark INT CHECK (mark BETWEEN 0 AND 100)
);
```

Only marks within the range 0–100 are accepted.

5. PRIMARY KEY Constraint

Defines a unique identifier for table records.
Automatically applies UNIQUE + NOT NULL.

```
CREATE TABLE department (
    dept_id CHAR(3) PRIMARY KEY,
    dept_name VARCHAR(50)
);
```

A table can have **only one** primary key (which may contain multiple columns).

6. FOREIGN KEY Constraint

Links a column in one table to the **primary key** in another table to maintain referential integrity.

```
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    dept_id CHAR(3),
    FOREIGN KEY (dept_id) REFERENCES department(dept_id)
);
```

Now, every employee's dept_id must exist in the department table.

7. AUTO_INCREMENT Constraint

Automatically generates sequential values for each new row.
Used mainly with primary keys.

```
CREATE TABLE customer (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50)
);
```

Each new insert increases customer_id automatically:
1, 2, 3, …

Example:

```
INSERT INTO customer (name) VALUES ('Kavya'), ('Arjun');
SELECT * FROM customer;
```

Output:

customer_id	name
1	Kavya
2	Arjun

8. ENUM Constraint

Restricts a column to a fixed list of possible values.

```
CREATE TABLE product (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(50),
    category ENUM('Electronics', 'Clothing', 'Food') NOT NULL
);
```

Attempting to insert any value outside the list will result in an error.

Example:

```
INSERT INTO product VALUES (1, 'TV', 'Electronics');  -- valid
INSERT INTO product VALUES (2, 'Shirt', 'Fashion');   -- invalid
```

9. INDEX (Non-Integrity Constraint)

An **index** is not a rule constraint but is used to improve query performance by speeding up searches.

```
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    salary DECIMAL(10,2)
);
```

```
CREATE INDEX idx_name ON employee(emp_name);
```

Note:

- Indexes improve performance for read-heavy operations.
- Too many indexes can slow down INSERT, UPDATE, and DELETE.

10. ON DELETE / ON UPDATE (Referential Actions)

Defines what happens to child table records when related parent records are updated or deleted.

Syntax

```
FOREIGN KEY (column)
REFERENCES parent_table (column)
ON DELETE action
ON UPDATE action;
```

Possible Actions

Action	Description

CASCADE	Propagates the change to dependent rows.
SET NULL	Sets child foreign key to NULL.
SET DEFAULT	Sets to a default value (not supported in all DBs).
NO ACTION	Prevents the change if dependent data exists.
RESTRICT	Similar to NO ACTION (MySQL default).

Example

```
CREATE TABLE department (
    dept_id CHAR(3) PRIMARY KEY,
    dept_name VARCHAR(50)
);

CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    dept_id CHAR(3),
    FOREIGN KEY (dept_id)
    REFERENCES department(dept_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

Effect:
If a department is deleted, all employees under that department are automatically removed.

11. Composite Constraints

A **composite primary key** or **unique constraint** uses multiple columns together.

CREATE TABLE enrollment (
  student_id INT,
  course_id INT,
  PRIMARY KEY (student_id, course_id)
);

Each pair of student_id and course_id must be unique.

Adding Constraints After Table Creation

ALTER TABLE student
ADD CONSTRAINT check_mark CHECK (mark BETWEEN 0 AND 100);

ALTER TABLE student
ADD CONSTRAINT unique_email UNIQUE (email);

Removing Constraints

```
ALTER TABLE student DROP CONSTRAINT check_mark;
ALTER TABLE student DROP PRIMARY KEY;
ALTER TABLE student DROP FOREIGN KEY fk_student_dept;
```

(Use SHOW CREATE TABLE student; to find actual constraint names.)

Hands-On Practice (MySQL)

1. **Create a Full Example Schema**

```
CREATE DATABASE constraint_enhanced;
USE constraint_enhanced;

CREATE TABLE department (
    dept_id CHAR(3) PRIMARY KEY,
    dept_name VARCHAR(50) UNIQUE
);

CREATE TABLE employee (
    emp_id INT PRIMARY KEY AUTO_INCREMENT,
    emp_name VARCHAR(50) NOT NULL,
    gender ENUM('Male','Female','Other') NOT NULL,
    salary DECIMAL(10,2) CHECK (salary > 0),
    dept_id CHAR(3),
    FOREIGN KEY (dept_id)
        REFERENCES department(dept_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE INDEX idx_emp_name ON employee(emp_name);
```

2. **Insert Valid Data**

```
INSERT INTO department VALUES ('D01','HR'), ('D02','IT');
```

```
INSERT INTO employee (emp_name, gender, salary, dept_id)
VALUES ('Meera','Female',55000,'D01'),
       ('Arjun','Male',70000,'D02');
```

3. **Test Referential Action**

```
DELETE FROM department WHERE dept_id='D01';
SELECT * FROM employee;  -- Employee Meera also deleted automatically
```

Checkpoint Review

1. What is the purpose of constraints in a database?
2. How does AUTO_INCREMENT simplify key management?
3. Explain the difference between CASCADE and RESTRICT in a foreign key.
4. What is an ENUM, and when should it be used?
5. Write SQL to add a CHECK constraint ensuring salary > 0 after table creation.

Reflection Prompt

Imagine you're designing a banking database.
How would you use constraints to ensure:

- Each account number is unique,
- Balances cannot be negative,
- Deleting a customer deletes all their accounts automatically?

Summary

- Constraints enforce data integrity and control valid input.
- Core constraints: NOT NULL, UNIQUE, DEFAULT, CHECK, PRIMARY KEY, FOREIGN KEY.
- Advanced constraints: AUTO_INCREMENT, ENUM, INDEX, and ON DELETE/UPDATE actions.
- Constraints can be defined during or after table creation.
- Proper constraint design maintains accuracy, consistency, and referential integrity across relational databases.

2.6 SQL Functions

Subtopics

- Numeric Functions
- String Functions

- Date and Time Functions
- Aggregate Functions
- Conversion Functions
- GROUP BY, HAVING, ORDER BY

Learning Outcome

After completing this topic, you should be able to:

- Use SQL's built-in functions to calculate, manipulate, and format data.
- Apply aggregate functions to summarize results.
- Group, filter, and order query results efficiently.

Concept Explanation

Functions in SQL perform operations on data and return a value.
They're classified into:

- **Single-row functions:** act on each row and return one result per row (e.g., ROUND(), LCASE(), NOW()).
- **Aggregate functions:** operate on groups of rows and return one result for the group (e.g., COUNT(), SUM()).

1. Numeric Functions

Used to perform calculations on numeric data.

Function	Description	Example	Result
ABS(x)	Returns absolute value	ABS(-8)	8
ROUND(x,d)	Rounds to d decimal places	ROUND(45.678,2)	45.68
CEIL(x)	Smallest integer ≥ x	CEIL(10.2)	11
FLOOR(x)	Largest integer ≤ x	FLOOR(10.9)	10
POWER(x,y)	x raised to y	POWER(2,3)	8
MOD(x,y)	Remainder of division	MOD(11,3)	2

Example Query

```
SELECT emp_id, salary, ROUND(salary * 1.05, 2) AS revised_salary
FROM employee;
```

2. String Functions

Used to process and manipulate text data.

Function	Purpose	Example	Output
LENGTH(str)	Returns number of characters	LENGTH('SQL')	3
LOWER(str)	Converts to lowercase	LOWER('HELLO')	hello
UPPER(str)	Converts to uppercase	UPPER('hello')	HELLO
CONCAT(a,b)	Combines two strings	CONCAT('Data','Base')	Database
SUBSTRING(str, start, len)	Extracts substring	SUBSTRING('Database',1,4)	Data
REPLACE(str, find, replace)	Replaces text	REPLACE('SQL Basics','Basics','Core')	SQL Core
LTRIM(str) / RTRIM(str)	Trims spaces	LTRIM(' test')	test

Example Query

```
SELECT emp_name,
       CONCAT(UPPER(emp_name), ' - ', dept_id) AS display_name
FROM employee;
```

3. Date and Time Functions

Used for handling and formatting date or time data.

Function	Purpose	Example	Output (Example)
CURDATE()	Returns current date	2025-10-07	
CURTIME()	Returns current time	14:25:00	
NOW()	Current date and time	2025-10-07 14:25:00	
YEAR(date)	Extracts year	YEAR('2025-10-07') → 2025	
MONTHNAME(date)	Returns month name	MONTHNAME('2025-10-07') → October	
DATEDIFF(d1,d2)	Difference in days	DATEDIFF('2025-10-10','2025-10-07') → 3	
DATE_ADD(date, INTERVAL n DAY)	Adds days	DATE_ADD('2025-10-07', INTERVAL 5 DAY) → 2025-10-12	

Example Query

```
SELECT emp_name, DATE_ADD(NOW(), INTERVAL 30 DAY) AS review_date
FROM employee;
```

4. Aggregate Functions

Operate on groups of rows and return a single value.

Function	Purpose	Example
COUNT(column)	Number of rows	COUNT(*)
SUM(column)	Total sum	SUM(salary)
AVG(column)	Average value	AVG(mark)
MIN(column)	Smallest value	MIN(salary)

MAX(column)	Largest value	MAX(salary)

Example Query

```
SELECT department,
       COUNT(emp_id) AS total_employees,
       AVG(salary) AS avg_salary
FROM employee
GROUP BY department;
```

5. Conversion Functions

Convert data types or formats.

Function	Purpose	Example	Result
CAST(expr AS type)	Converts to specified type	CAST('123' AS INT)	123
CONVERT(expr, type)	Similar to CAST	CONVERT('2025-10-07', DATE)	2025-10-07
FORMAT(number, d)	Formats number with decimals	FORMAT(1234.5, 2)	1,234.50

6. GROUP BY Clause

```
Groups rows that have the same values into summary rows, often with
aggregate functions.
SELECT department, COUNT(emp_id) AS total
FROM employee
GROUP BY department;
```

7. HAVING Clause

Filters grouped results — used with GROUP BY.
Unlike WHERE, which filters individual rows, HAVING filters aggregated groups.

```
SELECT department, AVG(salary) AS avg_salary
FROM employee
GROUP BY department
HAVING AVG(salary) > 60000;
```

8. ORDER BY Clause

Sorts query output in ascending (ASC) or descending (DESC) order.

```
SELECT emp_name, salary
FROM employee
ORDER BY salary DESC;
```

Hands-On Practice (MySQL)

1. **Create Sample Table**

```
CREATE DATABASE functions_demo;
USE functions_demo;

CREATE TABLE sales (
    sale_id INT PRIMARY KEY AUTO_INCREMENT,
    product VARCHAR(50),
    quantity INT,
    price DECIMAL(10,2),
    sale_date DATE
);

INSERT INTO sales (product, quantity, price, sale_date) VALUES
('Laptop', 3, 55000, '2025-09-01'),
('Mouse', 10, 700, '2025-09-02'),
('Keyboard', 5, 1500, '2025-09-05'),
('Laptop', 2, 56000, '2025-09-10');
```

2. **Use Aggregate and Group Functions**

```
SELECT product,
       SUM(quantity) AS total_units,
       AVG(price) AS avg_price,
       SUM(quantity * price) AS total_revenue
FROM sales
GROUP BY product
HAVING SUM(quantity) > 2
ORDER BY total_revenue DESC;
```

3. **Apply String and Date Functions**

```
SELECT product,
```

```
       UPPER(product) AS upper_name,
       DATE_ADD(sale_date, INTERVAL 10 DAY) AS next_sale_day
FROM sales;
```

Checkpoint Review

1. What is the difference between single-row and aggregate functions?
2. Write a query to find the total revenue for each product.
3. How does HAVING differ from WHERE?
4. Which function will convert text to uppercase?
5. Write SQL to display employees with average salary greater than 60 000.

Reflection Prompt

Think of a “Sales” dashboard.
How would you use SQL functions to show total sales, average order value, and monthly trends?

Summary

- SQL functions automate calculations and data formatting.
- **Numeric, String, Date, Conversion** → single-row functions.
- **Aggregate** → summarize groups of rows.
- GROUP BY and HAVING manage grouped data.
- ORDER BY arranges results for presentation.
- Together, these features turn raw data into meaningful information.

2.7 Clauses & Conditions

Subtopics

- WHERE vs HAVING
- Logical Operators (AND, OR, NOT)
- Wildcards and Pattern Matching (LIKE, %, _)

Learning Outcome

By the end of this topic, you will be able to:

- Understand how to filter data using logical operators.
- Differentiate between WHERE and HAVING clauses.
- Use wildcard characters for flexible pattern-based searching.

Concept Explanation

Filtering is one of the most common and essential operations in SQL.
It allows you to **retrieve only the data that meets specific criteria**, improving query efficiency and accuracy.

SQL provides multiple clauses and logical operators to achieve precise filtering:

- WHERE → filters individual rows before grouping.
- HAVING → filters grouped results after aggregation.
- LIKE, IN, BETWEEN, and logical operators like AND, OR, NOT refine filtering further.

1. WHERE vs HAVING Clause

Feature	WHERE	HAVING
Used With	Individual rows	Groups (after aggregation)
Filters	Before grouping	After grouping
Can Use Aggregate Functions?	❌ No	✅ Yes
Used With	SELECT, UPDATE, DELETE	SELECT (with GROUP BY)

Example 1 – Using WHERE

```
SELECT emp_name, department, salary
FROM employee
WHERE salary > 60000;
```

✅ Filters rows where salary exceeds 60,000.

Example 2 – Using HAVING

```
SELECT department, AVG(salary) AS avg_salary
FROM employee
GROUP BY department
HAVING AVG(salary) > 60000;
```

✅ Filters *groups* where the average salary exceeds 60,000.

2. Logical Operators

Logical operators combine multiple conditions in a query.

a. AND Operator

All conditions must be true.

```
SELECT * FROM employee
```

```
WHERE department = 'IT' AND salary > 60000;
```

✅ Returns employees from IT with salary > 60,000.

b. OR Operator

At least one condition must be true.

```
SELECT * FROM employee
WHERE department = 'HR' OR department = 'Finance';
```

✅ Returns employees from either HR or Finance.

c. NOT Operator

Negates a condition.

```
SELECT * FROM employee
WHERE NOT department = 'IT';
```

✅ Returns all employees except those in IT.

d. Combining Operators

You can combine them with parentheses to control precedence.

```
SELECT * FROM employee
WHERE (department = 'IT' OR department = 'Finance') AND salary > 50000;
```

Operator Precedence

SQL evaluates conditions in this order:

1. NOT
2. AND
3. OR

3. Wildcards and Pattern Matching

The LIKE operator allows flexible filtering using **wildcards** for text patterns.

Symbol	Meaning	Example	Matches
%	Any sequence of characters	A%'	Arjun, Anita

_	A single character	A_'	An, Aj
[ ]	Any character within brackets (in SQL Server)	[ABC]%	Alice, Bob, Charles
[^]	Excludes characters (in SQL Server)	[^A]%	names not starting with A

Example 1 – Starting with a Letter

```
SELECT emp_name
FROM employee
WHERE emp_name LIKE 'A%';
```

Example 2 – Ending with a Character

```
SELECT emp_name
FROM employee
WHERE emp_name LIKE '%a';
```

Example 3 – Containing Substring

```
SELECT emp_name
FROM employee
WHERE emp_name LIKE '%ee%';
```

Example 4 – Exact Character Count

```
SELECT emp_name
FROM employee
WHERE emp_name LIKE 'A__';
```

✅ Returns all 3-letter names starting with A.

4. Pattern Matching with NOT LIKE

Filters records not matching the specified pattern.

```
SELECT emp_name
FROM employee
WHERE emp_name NOT LIKE 'A%';
```

✅ Displays all employees whose names do not start with A.

5. Combining Wildcards with Logical Operators

```
SELECT emp_name, department
FROM employee
```

```
WHERE (emp_name LIKE 'A%' OR emp_name LIKE 'K%')
  AND department = 'IT';
```

✅ Returns employees from IT whose names start with A or K.

Hands-On Practice (MySQL)

1. **Create Sample Table**

```
CREATE DATABASE clauses_demo;
USE clauses_demo;

CREATE TABLE employee (
    emp_id INT PRIMARY KEY AUTO_INCREMENT,
    emp_name VARCHAR(50),
    department VARCHAR(30),
    salary DECIMAL(10,2)
);

INSERT INTO employee (emp_name, department, salary) VALUES
('Arjun', 'IT', 72000),
('Meera', 'HR', 50000),
('Kavya', 'Finance', 64000),
('Divya', 'IT', 68000),
('Rahul', 'Sales', 55000);
```

2. **Use WHERE and Logical Operators**

```
SELECT * FROM employee
WHERE (department = 'IT' OR department = 'Finance')
  AND salary > 60000;
```

3. **Use GROUP BY and HAVING**

```
SELECT department, AVG(salary) AS avg_salary
FROM employee
GROUP BY department
HAVING AVG(salary) > 60000;
```

4. **Pattern Matching**

```
SELECT emp_name FROM employee WHERE emp_name LIKE '%ya';
```

Checkpoint Review

1. What is the key difference between WHERE and HAVING?
2. How would you find all employees not from the “IT” department?
3. What does the query emp_name LIKE 'A%' return?
4. How does SQL handle AND, OR, and NOT precedence?
5. Write a query to get departments where average salary > 60,000.

Reflection Prompt

If you were building a student database, how could you use pattern matching to find all students whose names start with “S” and belong to “CSE”?

Summary

- WHERE filters individual rows; HAVING filters grouped results.
- Logical operators (AND, OR, NOT) combine multiple conditions.
- LIKE and wildcards enable flexible text filtering.
- Use parentheses to control operator precedence.
- Effective use of conditions makes queries more targeted and efficient.