National University
Of Computer and Emerging Sciences

Foundation of Advancement
Of Science and Technology

Lab Number:	03
Course Code:	CL2005
Course Title:	Database Systems - Lab
Semester:	Spring 2026
Agenda:	SQL Basics (Cont.) DQL
Instructor:	Muhammad Mehdi
Email:	muhammad.mehdi@nu.edu.pk
Office:	Rafaqat Lab

Table of Contents

1 SQL Command Categories 1
2 SQL Data Query Language (DQL) 1
2.1 Selecting All Columns 1
2.2 Selecting Specific Columns 2
2.3 Selecting Distinct Values 2
2.4 The WHERE Clause 3
2.5 Comparison Operators 3
2.6 Logical Operators 4
2.7 Special Filtering Operators 5
2.7.1 BETWEEN 5
2.7.2 IN 6
2.7.3 LIKE 6
2.8 Ordering and Limiting Results 7
2.8.1 ORDER BY 7
2.8.2 LIMIT 8
2.8.3 OFFSET 9
2.9 Arithmetic Expressions in SELECT Statements 9
2.10 Complete SQL Query Structure 10
3 Aggregate Functions 10
3.1 COUNT() 11
3.2 SUM() 12
3.3 AVG() 12
3.4 MIN() 12
3.5 MAX() 13
4 Column Aliases 13
5 Conditional Expressions in SELECT Statements 14
5.1 Basic Syntax 14
5.2 Examples 15
5.2.1 Grading System 15
5.2.2 Categorizing Values 15

1 SQL Command Categories

SQL commands are generally divided into categories based on their purpose.

Main categories:

- DDL (Data Definition Language)
- DML (Data Manipulation Language)
- DQL (Data Query Language)
- TCL (Transaction Control Language)
- DCL (Data Control Language)

These SQL commands allow us to:

- define database structures
- insert, update, and delete data
- retrieve data using queries
- And more ...

2 SQL Data Query Language (DQL)

DQL commands query and retrieve the different sets of data from the database. `SELECT` keyword is used to retrieve data from one or more tables.

2.1 Selecting All Columns

The `SELECT *` statement retrieves **all columns and all rows** from a table. It is useful when you want to view the complete dataset without specifying individual columns.

General Syntax:

```
SELECT * FROM table_name;
```

Example:

```
SELECT * FROM students;
```

2.2 Selecting Specific Columns

Sometimes, all the columns in a table are not needed. The SELECT statement allows you to specify only the columns you want to display.

General Syntax:

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

Example:

```
SELECT name, cgpa
FROM students;
```

2.3 Selecting Distinct Values

The DISTINCT keyword is used to **remove duplicate rows** from the result set. It ensures that only unique values are returned for the specified column(s).

It applies to **all selected columns together**, not individually. If multiple columns are specified, the combination of values must be unique.

General Syntax:

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

Examples:

```
-- Returns each unique name value from the students table
SELECT DISTINCT name
FROM students;

-- Rows are considered duplicates where both name & cgpa are same
SELECT DISTINCT name, cgpa
FROM students;
```

2.4 The WHERE Clause

The `WHERE` clause is used to filter rows based on a condition. Only rows that satisfy the condition are included in the results.

In SQL, every condition may evaluate to one the following values:

- TRUE
- FALSE
- **UNKNOWN** (in case of `NULL`/missing values)

General Syntax:

```
SELECT columns
FROM table_name
WHERE condition;
```

Example:

```
SELECT *
FROM students
WHERE department = 'CS';
```

2.5 Comparison Operators

Comparison operators are used in the `WHERE` clause to **compare column values with specified conditions**. They help filter rows based on numeric, text, or date comparisons.

Operator	Meaning
`=`	equal
`!= or <>`	not equal
`>`	greater than
`<`	less than
`>=`	greater than or equal

`<=`	less than or equal
`IS NULL`	check for missing value
`IS NOT NULL`	returns true if non-null value exists

Examples:

```
-- Select students with cgpa equal to 2.0
SELECT *
FROM students
WHERE cgpa = 2.0;

-- Select students whose department is not CS
SELECT *
FROM students
WHERE department <> 'CS';

-- Select students with cgpa greater than 3.0
SELECT *
FROM students
WHERE cgpa > 3.0;

-- Select students who do not have a department
SELECT *
FROM students
WHERE department IS NULL;

-- Select students who have an email address
SELECT *
FROM students
WHERE email IS NOT NULL;
```

2.6 Logical Operators

Logical operators are used to **combine multiple conditions** in a `WHERE` clause. They allow more complex filtering logic.

Operator	Meaning
AND	all conditions must be true
OR	at least one condition must be true
NOT	negates condition

Examples:

```
-- Select students from CS department with cgpa greater than 3.5
SELECT *
FROM students
WHERE department = 'CS' AND cgpa > 3.5;

-- Select students from CS or IT department
SELECT *
FROM students
WHERE department = 'CS' OR department = 'IT';

-- Select students who are not from SE department
SELECT *
FROM students
WHERE NOT department = 'SE';
```

2.7 Special Filtering Operators

SQL provides **special operators** that make filtering more flexible and concise. Three commonly used operators are:

1. BETWEEN
2. IN
3. LIKE

2.7.1 BETWEEN

It is used to filter values **within a specific range**. It is inclusive, meaning it includes both the lower and upper bounds.

General Syntax:

```
column_name BETWEEN lower_value AND upper_value
```

Example:

```
-- Find students whose 3.5 <= cgpa <= 2.0
SELECT *
FROM students
WHERE cgpa BETWEEN 3.5 AND 2.0;

/* This is equivalent to writing:
WHERE cgpa >= 3.5 AND cgpa <= 2.0; */
```

2.7.2 IN

It allows you to filter rows based on **a list of possible values**. It is a shorthand for multiple `OR` conditions.

General Syntax:

```
column_name IN (value1, value2, value3, ...)
```

Example:

```
-- Find students whose name is Ahmad, Ali, or Aslam
SELECT *
FROM students
WHERE name IN ('Ahmad', 'Ali', 'Aslam');

/* This is equivalent to writing:
WHERE name = 'Ahmad' OR name = 'Ali' OR name = 'Aslam'; */
```

2.7.3 LIKE

It is used for **pattern matching** in text columns. It allows you to search for values that match a specific pattern using **wildcards**.

`LIKE` is **case-insensitive** for ASCII letters by default in SQLite.

Wildcard	Meaning

%	matches zero or more characters
_	matches exactly one character

General Syntax:

```
column_name LIKE pattern
```

Examples:

```
-- Find all students whose email contains "gmail"
SELECT *
FROM students
WHERE email LIKE '%gmail%';


-- Find all students whose name starts with 'A'
SELECT *
FROM students
WHERE name LIKE 'A%';


-- Find students whose name is exactly 4 characters long ending with
'i'
SELECT *
FROM students
WHERE name LIKE '___i';
```

2.8 Ordering and Limiting Results

2.8.1 ORDER BY

`ORDER BY` sorts query results in ascending or descending order based on the specified column(s). The default sorting order is ascending.

General Syntax:

```
SELECT columns
```

```
FROM table_name
ORDER BY column1 ASC|DESC, column2 ASC|DESC, ...;
```

Examples:

```
-- Single column sort (ranking)
SELECT *
FROM students
ORDER BY cgpa DESC;

/* Multi-column sort: primary sort (grouping), secondary sort
(ranking within group) */
SELECT *
FROM students
ORDER BY department ASC, cgpa DESC;
```

2.8.2 LIMIT

`LIMIT` restricts the **maximum number of rows** returned by a query.
It is commonly used for:

- Previewing data
- Fetching top-N results
- Pagination

It is often used together with `ORDER BY` to guarantee specific order in the rows first before limiting.

General Syntax:

```
SELECT columns
FROM table_name
LIMIT number;
```

Example:

```
-- Returns at most 3 rows from the students table.
SELECT *
FROM students
```

```
LIMIT 3;
```

2.8.3 OFFSET

`OFFSET` specifies the **number of rows to skip** before starting to return rows. Its numbering starts with **0 (default value)** instead of 1.

It is typically used together with `LIMIT`, especially for **pagination**.

General Syntax:

```
SELECT columns
FROM table_name
LIMIT number
OFFSET number_to_skip;
```

Example:

```
-- Skips the first 2 rows and then returns the next 3 rows.
SELECT *
FROM students
LIMIT 3
OFFSET 2;
```

2.9 Arithmetic Expressions in SELECT Statements

SQL allows arithmetic expressions to be used inside `SELECT` statements. These expressions can perform calculations using column values and constants.

Supported operators:

- Addition (`+`)
- Subtraction (`-`)
- Multiplication (`*`)
- Division (`/`)

Arithmetic expressions are evaluated row by row. Arithmetic operations result in a **dynamically calculated/computed column**. It can be useful for:

- calculating percentages
- computing totals
- transforming raw values
- normalizing data

General Syntax:

```
SELECT arithmetic_expressions, ...
FROM table_name
```

Example:

```
-- Displays the name and expected graduation year of the students
SELECT name, (8 - current_semester) / 2
FROM students;
```

2.10 Complete SQL Query Structure

The following order of keywords is required for using the different query commands together in a single statement.

General Syntax:

```
SELECT columns
FROM table_name
WHERE condition
ORDER BY columns ASC|DESC
LIMIT number
OFFSET number_to_skip;
```

Example:

```
/* Retrieves the student with the 3rd highest CGPA among those
who have a non-null email address */

SELECT *
FROM students
WHERE email IS NOT NULL
```

```
ORDER BY cgpa DESC
LIMIT 1
OFFSET 2;
```

3 Aggregate Functions

Aggregate functions operate on multiple rows and return a single summarized value. They are commonly used for reporting and analysis tasks.

Aggregate functions ignore NULL values by default (except COUNT(*)). These functions are applied to a column and produce one result for the entire table when GROUP BY is not used.

3.1 COUNT()

It returns the number of rows in a table. It is used to measure the size of datasets, number of records, or availability of data.

It can be used for two purposes:

- Counting the total number of records in a table
- Counting the number of non-null values in a column

General Syntax:

```
-- Counts all rows, including rows containing NULL values
SELECT COUNT(*)
FROM table_name;

-- Counts only non-null values in the given column
SELECT COUNT(column_name)
FROM table_name;
```

Examples:

```
-- Returns the total number of records in students table
SELECT COUNT(*)
FROM students;
```

```
-- Returns the total number of non-null departments
SELECT COUNT(department)
FROM students;

-- Returns the total number of unique departments
SELECT COUNT(DISTINCT department)
FROM students;

-- Counts the total number of students with cgpa > 3.5
SELECT COUNT(roll_number)
FROM students
WHERE cgpa > 3.5;
```

3.2 SUM()

It returns the total of all numeric values in a column. This is useful for calculating totals such as total marks, total salary expense, or total credits.

SUM() works only on numeric data types.

Example:

```
-- Returns the sum of all credit hours in the column
SELECT SUM(credit_hours) FROM courses;
```

3.3 AVG()

It returns the average (mean) of numeric values. AVG() ignores NULL values. It is often used to calculate average grades, scores, or ratings.

Example:

```
-- Calculates the average CGPA
SELECT AVG(cgpa) FROM students;
```

3.4 MIN()

It returns the smallest value in a column. This can be used to find minimum salary, lowest score, or earliest date.

Example:

```
-- Calculates the lowest CGPA
SELECT MIN(cgpa) FROM students;
```

3.5 MAX()

It returns the largest value in a column. This can be used to find top performers, maximum salary, or latest date.

Example:

```
-- Calculates the highest CGPA
SELECT MAX(cgpa) FROM students;
```

4 Column Aliases

Column aliases allow you to rename output columns in query results. They improve readability and presentation without changing the actual table schema.

Aliases are especially useful when:

- performing calculations
- displaying derived values
- presenting reports
- avoiding long or unclear column names

Aliases only affect the query output, not the database structure.

General Syntax:

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

Example:

```
SELECT MIN(cgpa) AS lowest_cgpa
FROM students;

SELECT name AS student_name, cgpa AS gpa
FROM students;
```

5 Conditional Expressions in SELECT Statements

The CASE statement in SQL is used to apply **conditional logic** in queries. It works like an **if-else statement** or a **switch-case** statement in programming languages.

The CASE expression does not change the actual data in the table. It only changes how the data is displayed in the query result.

It allows you to:

- Create new calculated columns
- Categorize data
- Convert values into readable labels
- Perform conditional counting and calculations

5.1 Basic Syntax

The CASE expression is composed of the following keywords:

1. **WHEN:** Each WHEN checks a condition
2. **THEN:** If the condition is true, the corresponding THEN value is returned
3. **ELSE:** It is optional (if omitted and no condition matches, result is NULL)

```
-- For if-else like statements
CASE
  WHEN condition1 THEN result1
  WHEN condition2 THEN result2
  ELSE result3
END
```

```
-- Short-form: For switch-case like statements
CASE column
  WHEN value1 THEN result1
  WHEN value2 THEN result2
  ELSE result3
END
```

5.2 Examples

5.2.1 Grading System

```
-- Creates a new column grade based on specified conditions
SELECT
  name,
  department,
  CASE
    WHEN cgpa >= 3.67 THEN 'A'
    WHEN cgpa >= 3.33 THEN 'B'
    WHEN cgpa >= 3.0 THEN 'C'
    ELSE 'F'
  END AS grade
FROM students;
```

5.2.2 Categorizing Values

```
-- Creates a new column performance_label using short-form case
SELECT
  *,
  CASE cgpa
    WHEN 4.00 THEN 'Perfect GPA'
    WHEN 3.50 THEN 'Excellent'
    WHEN 3.00 THEN 'Good'
    ELSE 'Needs Improvement'
  END AS performance_label
FROM students;
```