

Chapter 4 · Database Query using SQL

CBSE Class 12 Informatics Practices (065) · Unit 2 = 25 marks · Ultimate Revision Sheet

60-SECOND RECALL

Function families (4)	Math · Text · Date · Aggregate
5 aggregate functions	<code>MAX()</code> <code>MIN()</code> <code>AVG()</code> <code>SUM()</code> <code>COUNT()</code>
WHERE vs HAVING	WHERE filters rows BEFORE grouping · HAVING filters groups AFTER
COUNT(*) vs COUNT(col)	<code>COUNT(*)</code> counts all rows · <code>COUNT(col)</code> skips NULL
Clause order	SELECT → FROM → WHERE → GROUP BY → HAVING → ORDER BY
Equi-join	Join two tables on a common column using =

1. Math Functions

THE THREE PRESCRIBED

Function	What it does	Example → Result
<code>POWER(x, y)</code>	x raised to the power y	<code>POWER(2, 3)</code> → 8
<code>ROUND(x, d)</code>	Rounds x to d decimal places. A negative d rounds to the left of the decimal point.	<code>ROUND(15.678, 2)</code> → 15.68 <code>ROUND(15.678)</code> → 16 <code>ROUND(158, -1)</code> → 160
<code>MOD(x, y)</code>	Remainder when x is divided by y	<code>MOD(10, 3)</code> → 1

TRAP `ROUND` with a **negative** second argument rounds to tens, hundreds and so on. `ROUND(4567, -2)` gives **4600**. This exact case is a favourite 1-marker.

2. Text (String) Functions

ALL PRESCRIBED TEXT FUNCTIONS

Function	What it does	Example → Result
<code>UCASE()</code> / <code>UPPER()</code>	Converts to upper case	<code>UCASE('india')</code> → INDIA
<code>LCASE()</code> / <code>LOWER()</code>	Converts to lower case	<code>LCASE('INDIA')</code> → india
<code>MID()</code> / <code>SUBSTRING()</code> / <code>SUBSTR()</code>	Extracts a substring: (string, start, length). Position starts at 1 , not 0.	<code>MID('Informatics', 3, 4)</code> → form
<code>LENGTH()</code>	Number of characters, including spaces	<code>LENGTH('Data Base')</code> → 9
<code>LEFT(str, n)</code>	Leftmost n characters	<code>LEFT('Computer', 4)</code> → Comp
<code>RIGHT(str, n)</code>	Rightmost n characters	<code>RIGHT('Computer', 3)</code> → ter
<code>INSTR(str, sub)</code>	Position of the first occurrence; 0 if not found	<code>INSTR('Informatics', 'mat')</code> → 5
<code>LTRIM()</code>	Removes leading spaces	<code>LTRIM(' Hi')</code> → Hi
<code>RTRIM()</code>	Removes trailing spaces	<code>RTRIM('Hi ')</code> → Hi

TRIM()

Removes **both** leading and trailing spaces

TRIM(' Hi ') → Hi

- THE TWO TEXT TRAPS** 1. SQL string positions start at 1, not 0. MID('ABCDEF',1,3) gives ABC.
2. LENGTH() counts spaces too. LENGTH('A B') is 3.

3. Date Functions

ALL PRESCRIBED DATE FUNCTIONS

Function	What it does	Example (date = 2026-09-11)
NOW()	Current date AND time	2026-09-11 05:25:00
DATE()	Extracts the date part only	DATE(NOW()) → 2026-09-11
YEAR()	Year as a number	2026
MONTH()	Month as a number 1–12	9
MONTHNAME()	Month as a name	September
DAY()	Day of month as a number	11
DAYNAME()	Day as a name	Friday

TRAP MONTH() returns a **number**, MONTHNAME() returns a **word**. Same for DAY() and DAYNAME().
Read the question wording carefully — “month name” means MONTHNAME().

4. Aggregate Functions

THE FIVE

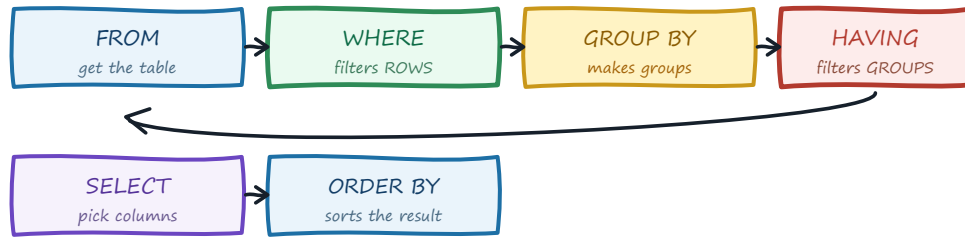
Function	Returns	NULL handling
MAX(col)	Largest value	Ignores NULL
MIN(col)	Smallest value	Ignores NULL
AVG(col)	Average	Ignores NULL
SUM(col)	Total	Ignores NULL
COUNT(col)	Number of non-NULL values	Ignores NULL
COUNT(*)	Total number of rows	Includes NULL rows

THE GUARANTEED 1-MARKER COUNT(*) counts every row. COUNT(column) skips NULLs.

If a table has 10 rows and 3 have NULL in Grade, then COUNT(*) = 10 but COUNT(Grade) = 7.
Also: all aggregate functions except COUNT(*) ignore NULL.

5. GROUP BY, HAVING and ORDER BY

You WRITE the clauses in one order. The database RUNS them in another.



WHERE runs BEFORE the grouping, HAVING runs AFTER it. That single fact decides which one the question wants.

- A condition on a plain column → WHERE e.g. Salary > 20000
- A condition on an AGGREGATE → HAVING e.g. COUNT(*) > 2

An aggregate inside WHERE is always an error — the groups do not exist yet at that point.

Written order is SELECT first; executed order is FROM first. HAVING can see aggregates only because it runs last.

THE CLAUSE ORDER — never change it

```
SELECT column, AGGREGATE(column)
FROM table
WHERE condition           -- filters ROWS, before grouping
GROUP BY column           -- makes the groups
HAVING condition          -- filters GROUPS, after grouping
ORDER BY column ASC|DESC; -- sorts the result
```

WORKED EXAMPLE

```
SELECT Dept, COUNT(*), AVG(Salary)
FROM Employee
WHERE Salary > 20000
GROUP BY Dept
HAVING COUNT(*) > 2
ORDER BY AVG(Salary) DESC;
```

Reads as: take employees earning over 20000 → group them by department → keep only departments with more than 2 such employees → sort by average salary, highest first.

WHERE vs HAVING — asked every year

Basis	WHERE	HAVING
Filters	Individual rows	Groups
When	BEFORE grouping	AFTER grouping
Aggregate functions	CANNOT be used	CAN be used
Needs GROUP BY	No	Usually yes

Rule: if the condition uses COUNT, SUM, AVG, MAX or MIN, it must go in HAVING.

```
ORDER BY Marks;           -- ASC is the DEFAULT
ORDER BY Marks DESC;
ORDER BY Dept, Marks DESC; -- sort by Dept, then Marks within it
```

6. Equi-Join — Working with Two Tables

DEFINITION An *equi-join* combines rows from two tables by matching a *common column* using the *equality (=)* operator. The common column is usually a *primary key* in one table and a *foreign key* in the other.

```
-- Method 1: WHERE clause
SELECT E.Name, D.DeptName
FROM   Employee E, Dept D
WHERE  E.DeptID = D.DeptID;

-- Method 2: JOIN ... ON (same result)
SELECT E.Name, D.DeptName
FROM   Employee E JOIN Dept D
ON     E.DeptID = D.DeptID;
```

THE RULES

1. Always state the *join condition*, or you get a *Cartesian product* (every row of A paired with every row of B).
2. If a column name exists in *both* tables, *qualify* it: E.DeptID, not just DeptID.
3. Cartesian product row count = *rows in A* × *rows in B*; column count = *columns in A* + *columns in B*.

TRAP If Employee has 5 rows / 4 columns and Dept has 3 rows / 2 columns, then a *Cartesian product* gives 15 rows and 6 columns. This calculation is a common 1–2 marker.

7. Revision — Core SQL from Class XI

DDL / DML / DQL

```
-- DDL: structure
CREATE DATABASE school;
USE school;
CREATE TABLE Student (RollNo INT PRIMARY KEY,
                        Name VARCHAR(30),
                        Marks DECIMAL(5,2));
ALTER TABLE Student ADD Grade CHAR(2);
DROP TABLE Student;

-- DML: data
INSERT INTO Student VALUES (1, 'Amit', 92.5);
UPDATE Student SET Marks = 95 WHERE RollNo = 1;
DELETE FROM Student WHERE RollNo = 1;

-- DQL: querying
SELECT * FROM Student;
SELECT DISTINCT Dept FROM Employee;
SELECT * FROM Student WHERE Marks BETWEEN 60 AND 90;
SELECT * FROM Student WHERE Name LIKE 'A%';
SELECT * FROM Student WHERE Grade IS NULL;
SELECT * FROM Student WHERE Dept IN ('Sales', 'HR');
```

Operator	Meaning
BETWEEN a AND b	Range, both ends INCLUSIVE
IN (list)	Matches any value in the list

LIKE '%'	% = any number of characters; _ = exactly one character
IS NULL / IS NOT NULL	Tests for NULL. Never use = NULL — it never matches.

TRAP NULL is not zero and not an empty string. It means “unknown”. You must use IS NULL ;
WHERE Grade = NULL returns no rows, always.

8. Predict the Output

Q1 SELECT ROUND(4567.789, -2); →
SELECT MOD(17, 5); →
SELECT POWER(3, 3); →

Q2 SELECT MID('Informatics', 3, 4); →
SELECT INSTR('Informatics', 'mat'); →
SELECT LENGTH('Data Base'); →

Q3 - table has 8 rows, 2 NULLs in Grade SELECT COUNT(*) →
SELECT COUNT(Grade) →

Q4 - A has 4 rows/3 cols, B has 2 rows/2 cols SELECT * FROM A, B; → (Cartesian product)

9. Task Triggers wording → which clause / function?

If the question says...	Use
"total number of records"	COUNT(*)
"how many employees have a grade recorded"	COUNT(Grade)
"department-wise average salary"	GROUP BY Dept + AVG()
"departments having more than 5 employees"	HAVING COUNT(*) > 5
"employees earning more than 30000"	WHERE Salary > 30000
"in descending order of marks"	ORDER BY Marks DESC
"names starting with A"	LIKE 'A%'
"second letter is a"	LIKE '_a%'
"salary between 20000 and 40000"	BETWEEN 20000 AND 40000
"where grade is not assigned"	IS NULL
"remove duplicate department names"	SELECT DISTINCT
"display name in capitals"	UCASE(Name)
"first 3 characters of the name"	LEFT(Name, 3) or MID(Name, 1, 3)
"month name of joining date"	MONTHNAME(DOJ)
"data from two tables together"	Equi-join with a matching condition
"add a new column to the table"	ALTER TABLE ... ADD
"change the marks of roll no 5"	UPDATE ... SET ... WHERE

MOST-ASKED BOARD QUESTIONS

- [4-5M]** Write SQL queries for the given table (usually 5 queries of 1 mark each).
- [2-3M]** Predict the output of the given SQL statements.
- [2M]** Differentiate between WHERE and HAVING.

4. [2M] Differentiate between `COUNT(*)` and `COUNT(column)`.
5. [2M] Write a query using an equi-join on two tables.
6. [1M] What is the default sort order of `ORDER BY`? → **ASC**

THE HABIT THAT SCORES — Write every query on **separate lines by clause** and end with a **semicolon**. Then check the order: `SELECT` → `FROM` → `WHERE` → `GROUP BY` → `HAVING` → `ORDER BY`. A clause out of order is marked wrong even if the logic is right.