

UNIT 6

Manipulating and Querying Data

(8 Hours) — Database Management System (CAC255) · BCA II Year / IV Semester

⚠ Important Note — Please Read

These notes were prepared quickly from the course syllabus and standard textbook knowledge. **They may contain errors, typing mistakes or incomplete points.** Please take care: cross-check definitions, formulas, SQL syntax and examples with your class notes, the prescribed textbook and your teacher before relying on them for exams or practicals.

Sample tables used in this unit (T-SQL syntax; MySQL differences are noted):

DeptID	DeptName
10	Sales
20	IT
30	HR

Department

EmpID	EName	Salary	DeptID
1	Ram	50000	10
2	Sita	60000	20
3	Hari	45000	20
4	Gita	70000	NULL

Employee

6.1 Adding Data with the INSERT Statement

INSERT forms

```
-- 1. Insert with column list (recommended)
INSERT INTO Employee (EmpID, EName, Salary, DeptID)
VALUES (1, 'Ram', 50000, 10);
-- 2. Insert without column list (values must follow table order)
INSERT INTO Employee VALUES (2, 'Sita', 60000, 20);
-- 3. Insert only some columns (others become NULL or DEFAULT)
INSERT INTO Employee (EmpID, EName) VALUES (5, 'Mina');
-- 4. Insert several rows at once
INSERT INTO Employee VALUES (3, 'Hari', 45000, 20),
                             (4, 'Gita', 70000, NULL);
```

```
-- 5. Insert rows copied from a query
INSERT INTO EmployeeBackup SELECT * FROM Employee WHERE Salary > 50000;
```

Note: Text and date values are written in single quotes. Identity columns must not be given a value unless IDENTITY_INSERT is ON.

6.2 Retrieving Data — SELECT, FROM and WHERE

SELECT statement — general form

```
SELECT column_list      -- what to show
FROM   table_name      -- from where
WHERE  condition        -- filter rows
GROUP BY columns       -- make groups
HAVING group_condition  -- filter groups
ORDER BY columns;      -- sort result
```

Basic SELECT examples

```
SELECT * FROM Employee;           -- all columns
SELECT EName, Salary FROM Employee; -- selected columns
SELECT EName AS Name, Salary * 12 AS Yearly FROM Employee; -- alias, expression
SELECT DISTINCT DeptID FROM Employee; -- remove duplicates
SELECT TOP 2 * FROM Employee;      -- T-SQL (MySQL: ... LIMIT 2)
```

Filtering with WHERE

Operator	Use	Example
=, <>, >, <, >=, <=	Comparison	WHERE Salary >= 50000
AND, OR, NOT	Combine conditions	WHERE DeptID = 20 AND Salary > 50000
BETWEEN ... AND ...	Inclusive range	WHERE Salary BETWEEN 45000 AND 60000
IN (list)	Match any value in the list	WHERE DeptID IN (10, 20)
LIKE	Pattern: % = any characters, _ = one character	WHERE EName LIKE 'S%'
IS NULL / IS NOT NULL	Test for NULL (never use = NULL)	WHERE DeptID IS NULL

Example 6.1 — Filtering

Display the name and salary of employees of department 20 who earn more than 50000.

Solution

SELECT EName, Salary FROM Employee WHERE DeptID = 20 AND Salary > 50000; →
Sita, 60000

6.3 Ordering and Grouping Data

ORDER BY (default is ASC)

```
SELECT * FROM Employee ORDER BY Salary DESC;           -- highest first
SELECT * FROM Employee ORDER BY DeptID ASC, Salary DESC; -- multiple keys
```

Aggregate (summary) functions

Function	Purpose	Example on Employee
COUNT(*)	Number of rows	4
COUNT(col)	Non-NULL values in a column	COUNT(DeptID) = 3
SUM(col)	Total	SUM(Salary) = 225000
AVG(col)	Average	AVG(Salary) = 56250
MAX(col) / MIN(col)	Largest / smallest	MAX = 70000, MIN = 45000

GROUP BY with HAVING

```
SELECT DeptID, COUNT(*) AS Total, AVG(Salary) AS AvgSal
FROM Employee
GROUP BY DeptID
HAVING COUNT(*) > 1;
```

DeptID	Total	AvgSal
20	2	52500

- **WHERE** filters rows *before* grouping; **HAVING** filters groups *after* aggregation.
- Every column in SELECT that is not inside an aggregate function must appear in GROUP BY.
- **Logical order of execution:** FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY.

6.4 Retrieving Data from Different Tables — Joins

A **join** combines rows from two or more tables based on a related column, normally a foreign key = primary key match.

INNER JOIN

Returns only rows that have matching values in both tables.

INNER JOIN

```
SELECT E.ENAME, D.DeptName
FROM   Employee E
INNER JOIN Department D ON E.DeptID = D.DeptID;
```

EName	DeptName
Ram	Sales
Sita	IT
Hari	IT

OUTER JOIN

LEFT, RIGHT and FULL joins

```
SELECT E.ENAME, D.DeptName FROM Employee E LEFT   OUTER JOIN Department D ON E.DeptID = D.DeptID
SELECT E.ENAME, D.DeptName FROM Employee E RIGHT  OUTER JOIN Department D ON E.DeptID = D.DeptID
SELECT E.ENAME, D.DeptName FROM Employee E FULL   OUTER JOIN Department D ON E.DeptID = D.DeptID
```

Join	Rows returned	Extra rows compared with INNER JOIN
LEFT OUTER	All rows of Employee + matches from Department	Gita — NULL
RIGHT OUTER	All rows of Department + matches from Employee	NULL — HR
FULL OUTER	All rows of both tables	Gita — NULL and NULL — HR

Note: MySQL does not support FULL OUTER JOIN directly; use LEFT JOIN UNION RIGHT JOIN.

CROSS JOIN

Returns the Cartesian product — every row of one table with every row of the other ($4 \times 3 = 12$ rows here). It has no ON clause.

CROSS JOIN

```
SELECT E.ENAME, D.DeptName FROM Employee E CROSS JOIN Department D;
```

SELF JOIN

A table is joined with itself using two different aliases, e.g. to find each employee's manager:

```
SELECT E.ENAME, M.ENAME AS Manager FROM Employee E JOIN Employee M ON  
E.MgrID = M.EmpID;
```

6.5 Building Nested Queries (Subqueries)

A **subquery** is a SELECT statement placed inside another statement, inside parentheses. The inner query runs first and passes its result to the outer query.

Types of subqueries

```
-- Scalar subquery: employees earning more than the average salary  
SELECT ENAME, Salary FROM Employee  
WHERE Salary > (SELECT AVG(Salary) FROM Employee);      -- Sita, Gita  
-- IN: employees who work in the IT department  
SELECT ENAME FROM Employee  
WHERE DeptID IN (SELECT DeptID FROM Department WHERE DeptName = 'IT');  
-- EXISTS: departments having at least one employee  
SELECT DeptName FROM Department D  
WHERE EXISTS (SELECT 1 FROM Employee E WHERE E.DeptID = D.DeptID);  
-- ALL / ANY: salary greater than every employee of dept 20  
SELECT ENAME FROM Employee  
WHERE Salary > ALL (SELECT Salary FROM Employee WHERE DeptID = 20);  -- Gita  
-- Subquery in FROM (derived table)  
SELECT T.DeptID, T.Total  
FROM (SELECT DeptID, COUNT(*) AS Total FROM Employee GROUP BY DeptID) AS T;
```

- **Correlated subquery:** the inner query uses a column of the outer query and is re-evaluated for each outer row (as in the EXISTS example).
- Subqueries can also be used with INSERT, UPDATE and DELETE.

6.6 UPDATE and DELETE

UPDATE and DELETE

```
-- Update one column of selected rows  
UPDATE Employee SET Salary = Salary * 1.10 WHERE DeptID = 20;  
-- Update several columns  
UPDATE Employee SET ENAME = 'Sita Sharma', DeptID = 10 WHERE EmpID = 2;  
-- Delete selected rows  
DELETE FROM Employee WHERE EmpID = 3;  
-- Delete all rows (structure remains)  
DELETE FROM Employee;
```

*Note: Forgetting the **WHERE** clause updates or deletes **every** row. Always test with a **SELECT** first. Foreign-key rules may reject deleting a parent row that is still referenced.*

6.7 Creating and Altering Views

A **view** is a **virtual table** defined by a stored **SELECT** query. It does not store data itself; it shows data from the base tables each time it is used.

Create, alter, drop view

```
CREATE VIEW ITEmployees AS
SELECT E.EmpID, E.EName, E.Salary
FROM   Employee E JOIN Department D ON E.DeptID = D.DeptID
WHERE  D.DeptName = 'IT';
SELECT * FROM ITEmployees;           -- use it like a table
ALTER VIEW ITEmployees AS           -- change the definition
SELECT EmpID, EName FROM Employee WHERE DeptID = 20;
DROP VIEW ITEmployees;              -- remove the view
```

- **Advantages:** simplifies complex queries, hides sensitive columns (security), gives logical data independence.
- **WITH CHECK OPTION** stops updates through the view that would make a row disappear from the view.
- Simple views on one table can be updated; views with joins, **GROUP BY** or **DISTINCT** usually cannot.
- MySQL uses **CREATE OR REPLACE VIEW** instead of **ALTER VIEW** in most cases.

Set operators

Operator	Result
UNION	Rows of both queries, duplicates removed (UNION ALL keeps duplicates)
INTERSECT	Rows common to both queries
EXCEPT (MINUS in Oracle)	Rows in the first query but not the second

Example 6.2 — Department-wise total salary

Display each department name with its total salary, only where the total is more than 50000, highest first.

Solution

Join + **GROUP BY** + **HAVING** + **ORDER BY**:

```
SELECT D.DeptName, SUM(E.Salary) AS TotalSal
FROM   Employee E INNER JOIN Department D ON E.DeptID = D.DeptID
GROUP BY D.DeptName
HAVING SUM(E.Salary) > 50000
ORDER BY TotalSal DESC;          -- IT 105000, Sales excluded (50000 is not > 50000)
```

Key Takeaways — Unit 6

- ✓ INSERT adds rows; UPDATE changes rows; DELETE removes rows — always use WHERE carefully.
- ✓ SELECT...FROM...WHERE filters rows; GROUP BY groups; HAVING filters groups; ORDER BY sorts.
- ✓ Aggregates: COUNT, SUM, AVG, MAX, MIN. Use IS NULL, not = NULL.
- ✓ INNER JOIN = matching rows only; LEFT/RIGHT/FULL OUTER keep unmatched rows; CROSS JOIN = Cartesian product.
- ✓ Subqueries can be scalar, IN, EXISTS, ANY/ALL, correlated or derived tables.
- ✓ A view is a stored query (virtual table); CREATE, ALTER and DROP VIEW manage it.

— End of Unit 6 —

Reminder: these notes may contain errors — verify with your textbook and teacher.