UNIT III · LESSON 04

SQL

DDL · constraints · DML · SELECT clauses · joins · subqueries · aggregates · views · triggers · PL/SQL basics

10
questions in Unit III
ER · Relational · Normal forms · SQL · Indexing

OVERVIEW SQL COMMAND FAMILIES

Family	Commands
DDL	CREATE, ALTER, DROP, TRUNCATE, RENAME
DML	INSERT, UPDATE, DELETE, MERGE
DQL	SELECT
DCL	GRANT, REVOKE
TCL	COMMIT, ROLLBACK, SAVEPOINT

- SQL = Structured Query Language (IBM **SEQUEL**, System R, 1970s); ANSI standard 1986
- Keywords are case-insensitive; strings in **single quotes**

WHERE OPERATORS

Operator	Example
BETWEEN (inclusive)	`sal BETWEEN 20000 AND 40000`
IN	`did IN (10, 20)`
LIKE	`name LIKE 'S%'` (%: any string, _ : one char)
IS NULL	`email IS NULL` (never `= NULL`)
AND / OR / NOT	logical

- `'_a%'` → second letter a; `'%n'` → ends with n

CREATE DDL & CONSTRAINTS

```
CREATE TABLE emp (
  eid    INT PRIMARY KEY,
  name   VARCHAR(30) NOT NULL,
  email  VARCHAR(40) UNIQUE,
  sal    DECIMAL(9,2) CHECK (sal > 0),
  did    INT DEFAULT 10,
  FOREIGN KEY (did) REFERENCES dept(did)
    ON DELETE CASCADE
);
ALTER TABLE emp ADD phone CHAR(10);
ALTER TABLE emp DROP COLUMN phone;
DROP TABLE emp;        -- table + data
TRUNCATE TABLE emp;    -- rows only
```

- Constraints: **NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT**

CREATE DDL & CONSTRAINTS (CONTD.)

- UNIQUE allows NULL(s); PRIMARY KEY = UNIQUE + NOT NULL, **one per table**

MODIFY DATA DML

```
INSERT INTO emp
  VALUES (1, 'Ravi', NULL, 50000, 20);
INSERT INTO emp (eid, name, sal)
  VALUES (2, 'Mala', 42000);
UPDATE emp SET sal = sal * 1.10
  WHERE did = 20;
DELETE FROM emp WHERE sal < 20000;
```

- `UPDATE` / `DELETE` without WHERE → **all rows**

ORDER OF EXECUTION SELECT ANATOMY

```
SELECT did, COUNT(*) AS n,
AVG(sal)
FROM emp
WHERE sal > 20000
GROUP BY did
HAVING COUNT(*) >= 3
ORDER BY n DESC;
```

Written order	Logical execution
SELECT	5
FROM	1
WHERE	2 (row filter)
GROUP BY	3
HAVING	4 (group filter)
ORDER BY	6

- `DISTINCT` removes duplicates; `AS` aliases; `LIMIT` / `TOP` / `FETCH FIRST` rows
- WHERE cannot use aggregates; HAVING can

GROUP FUNCTIONS AGGREGATES

Function	Note
COUNT(*)	counts all rows, including NULLs
COUNT(col)	ignores NULLs
SUM / AVG	ignore NULLs
MIN / MAX	any ordered type

- AVG(sal) = SUM(sal) / COUNT(sal), **not** COUNT(*)
- Non-aggregated SELECT columns must appear in GROUP BY

COMBINE RESULTS SET OPERATIONS

- `UNION` (removes duplicates), `UNION ALL` (keeps), `INTERSECT`, `EXCEPT` / `MINUS`
- Same number of columns, compatible types

COMBINE TABLES JOINS

Join	Returns
INNER JOIN	matching rows only
LEFT OUTER	all left rows + matches (NULL if none)
RIGHT OUTER	all right rows + matches
FULL OUTER	all rows of both
CROSS JOIN	Cartesian product m × n
NATURAL JOIN	equality on all same-named columns
SELF JOIN	table joined with itself (aliases)

```
SELECT e.name, d.dname
FROM emp e LEFT JOIN dept d ON e.did = d.did;
SELECT a.name, b.name AS manager
FROM emp a JOIN emp b ON a.mgr = b.eid;
```

NESTED SUBQUERIES

```
SELECT name FROM emp
WHERE sal > (SELECT AVG(sal) FROM emp);
SELECT name FROM emp e          -- correlated
WHERE sal > (SELECT AVG(sal) FROM emp
             WHERE did = e.did);
SELECT dname FROM dept d
WHERE NOT EXISTS (SELECT * FROM emp
                  WHERE did = d.did);
```

- `IN`, `NOT IN`, `EXISTS`, `ANY/SOME`, `ALL`
- `> ALL (…)` = greater than the maximum; `> ANY (…)` = greater than the minimum
- **Correlated** subquery runs once **per outer row**
- `NOT IN` with a NULL in the subquery → returns **no rows**

CLASSIC SECOND HIGHEST SALARY

```
SELECT MAX(sal) FROM emp
WHERE sal < (SELECT MAX(sal) FROM emp);
```

- N-th highest: `ORDER BY sal DESC LIMIT 1 OFFSET N-1` or `DENSE_RANK()`

VIRTUAL TABLE VIEWS

```
CREATE VIEW high_paid AS
  SELECT eid, name FROM emp
  WHERE sal > 60000;
```

- Stored query, no data (except **materialised** views)
- Security (hide columns), simplicity, logical independence
- Updatable only if simple (one table, no aggregates, no DISTINCT, no GROUP BY)
- `WITH CHECK OPTION` blocks updates that would leave the view

MORE DDL INDEXES, SEQUENCES, PRIVILEGES

- `CREATE INDEX ix_name ON emp(name);` speeds searches, slows updates
- `GRANT SELECT, UPDATE ON emp TO ravi WITH GRANT OPTION;`
- `REVOKE UPDATE ON emp FROM ravi;`
- `COMMIT` saves, `ROLLBACK` undoes to last commit, `SAVEPOINT s1` partial rollback point
- DDL statements cause an **implicit commit** (Oracle)

PL/SQL TRIGGERS & PROCEDURES

```
CREATE TRIGGER trg_sal
BEFORE UPDATE ON emp
FOR EACH ROW
BEGIN
  IF :NEW.sal < :OLD.sal THEN
    RAISE_APPLICATION_ERROR(-20001, 'cut');
  END IF;
END;
```

- **Trigger** = ECA rule (Event, Condition, Action); BEFORE/AFTER; row-level or statement-level
- **Stored procedure**: named block, called with EXEC/CALL; **function** returns a value
- **Cursor**: row-by-row processing (implicit / explicit: DECLARE, OPEN, FETCH, CLOSE)
- PL/SQL block: DECLARE … BEGIN … EXCEPTION … END
- **Embedded SQL** / dynamic SQL; JDBC, ODBC APIs

Q → A RAPID RECALL

Clause filtering groups	HAVING
Removes all rows, keeps structure	TRUNCATE

Q → A RAPID RECALL (CONTD.)

Pattern matching one character	`_`
Test for NULL	IS NULL
COUNT(*) vs COUNT(col)	COUNT(col) ignores NULLs

Q → A RAPID RECALL (CONTD.)

Join keeping all rows of both tables	FULL OUTER JOIN
Rows in CROSS JOIN of 5 and 4 rows	20
Duplicates kept by	UNION ALL
Executed once per outer row	correlated subquery
Command to give privileges	GRANT

TRAP EXAM TRAPS

- `WHERE col = NULL` is never true → use IS NULL
- `WHERE COUNT(*) > 2` is an error → HAVING
- BETWEEN includes both end values
- `DELETE` can be rolled back; `DROP` and `TRUNCATE` usually not
- `ORDER BY` is the only clause that can use a SELECT alias in standard SQL
- A view with GROUP BY is not updatable