SQL Interview Questions

1. How to find duplicates in a table.

   - students table: id column, name column and marks column.

     ```
     CREATE TABLE students(id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(20), marks INT);

     INSERT INTO students(name, marks) VALUES ('A', 70), ('B', 80), ('C', 90), ('A', 70), ('B', 85), ('A', 70);

     SELECT name, COUNT(name) FROM students
     GROUP BY name;
     ```

   - Duplicate using GROUP BY -- these names are duplicated SELECT name, COUNT(name) FROM students GROUP BY name HAVING COUNT(name) > 1;

     SELECT * FROM students WHERE name IN (SELECT name FROM students GROUP BY name HAVING COUNT(name) > 1);

     WITH dup_students AS (SELECT name FROM students GROUP BY name HAVING COUNT(name) > 1) SELECT * FROM students WHERE name IN (SELECT * FROM dup_students);

     SELECT name, COUNT(name), marks, COUNT(marks) FROM students GROUP BY name, marks HAVING COUNT(name) > 1 AND COUNT(marks) > 1;

     ```
     ```

   - Find duplicates using WINDOW functions.

     ```
     WITH studs AS(
         SELECT (ROW_NUMBER() OVER ()) sr, name, marks FROM students
     )
     SELECT * FROM studs;
     ```

```sql
WITH studs AS(
    SELECT (ROW_NUMBER() OVER (PARTITION BY name, marks)) sr, name, marks FROM students
)
SELECT * FROM studs
WHERE sr > 1;
```

2. How to delete duplicates from a table?

```sql
-- Find number of duplications
SELECT name, COUNT(name), marks, COUNT(marks) FROM students
GROUP BY name, marks
HAVING COUNT(name) > 1 AND COUNT(marks) > 1;

-- Display the repeated rows (Other than original/first row).
SELECT s1.* FROM students s1
INNER JOIN students s2 ON s1.id > s2.id
WHERE s1.name = s2.name AND s1.marks = s2.marks;

-- Delete the repeated rows (Other than original/first row).
DELETE s1 FROM students s1
INNER JOIN students s2 ON s1.id > s2.id
WHERE s1.name = s2.name AND s1.marks = s2.marks;
```

3. GROUP BY vs Window Functions

```sql
SELECT job, SUM(sal) FROM emp
GROUP BY job;

SELECT empno, ename, sal, job, SUM(sal) OVER(PARTITION BY job)
FROM emp;
```

```sql
SELECT ename, sal,
ROW_NUMBER() OVER () sr
FROM emp;

SELECT ename, sal,
ROW_NUMBER() OVER (ORDER BY sal) sr
FROM emp;

SELECT ename, sal,
ROW_NUMBER() OVER () sr,
RANK() OVER () rnk
FROM emp;

SELECT ename, sal,
ROW_NUMBER() OVER (ORDER BY sal) sr,
RANK() OVER (ORDER BY sal) rnk
FROM emp;

SELECT ename, sal,
ROW_NUMBER() OVER (ORDER BY sal) sr,
RANK() OVER (ORDER BY sal) rnk,
DENSE_RANK() OVER (ORDER BY sal) drnk
FROM emp;

SELECT ename, deptno, sal,
ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sal) sr,
RANK() OVER (PARTITION BY deptno ORDER BY sal) rnk,
DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal) drnk
FROM emp;
```

4. Window Specification in WINDOW functions.

```sql
SELECT ename, deptno, sal,
ROW_NUMBER() OVER (wnd) sr,
RANK() OVER (wnd) rnk,
DENSE_RANK() OVER (wnd) drnk
FROM emp
WINDOW wnd AS (PARTITION BY deptno ORDER BY sal);
```

5. Employees with Lowest/Highest Salaries

```sql
-- lowest sal in each dept
SELECT * FROM
(SELECT ename, deptno, sal,
ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sal) sr,
RANK() OVER (PARTITION BY deptno ORDER BY sal) rnk,
DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal) drnk
FROM emp) emp_ranks
WHERE rnk = 1;

-- second lowest sal in each dept
WITH emp_ranks AS (SELECT ename, deptno, sal,
ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sal) sr,
RANK() OVER (PARTITION BY deptno ORDER BY sal) rnk,
DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal) drnk
FROM emp)
SELECT * FROM emp_ranks
WHERE drnk = 2;

-- second lowest sal per dept
WITH emp_ranks AS (
SELECT ename, deptno, sal,
ROW_NUMBER() OVER (wnd) sr,
RANK() OVER (wnd) rnk,
DENSE_RANK() OVER (wnd) drnk
FROM emp
WINDOW wnd AS (PARTITION BY deptno ORDER BY sal)
)
SELECT * FROM emp_ranks
WHERE drnk = 2;

-- second highest sal per dept
WITH emp_ranks AS (
SELECT ename, deptno, sal,
ROW_NUMBER() OVER (wnd) sr,
RANK() OVER (wnd) rnk,
DENSE_RANK() OVER (wnd) drnk
FROM emp
WINDOW wnd AS (PARTITION BY deptno ORDER BY sal DESC)
)
SELECT * FROM emp_ranks
WHERE drnk = 2;

-- highest highest sal per dept
WITH emp_ranks AS (
SELECT ename, deptno, sal,
ROW_NUMBER() OVER (wnd) sr,
RANK() OVER (wnd) rnk,
DENSE_RANK() OVER (wnd) drnk
FROM emp
WINDOW wnd AS (PARTITION BY deptno ORDER BY sal DESC)
)
SELECT * FROM emp_ranks
WHERE drnk = 1;
```

6. Running Total -- Bank account mini-statement

```
SELECT accid, txdate, amount,
SUM(amount) OVER (PARTITION BY accid) run_bal
```

```
FROM transactions;

SELECT accid, txdate, amount,
SUM(amount) OVER (PARTITION BY accid ORDER BY txdate) run_bal
FROM transactions;

SELECT accid, txdate, amount,
SUM(amount) OVER (PARTITION BY accid ORDER BY txdate) run_bal
FROM transactions
WHERE accid = 1;

SELECT accid, txdate, amount,
SUM(amount) OVER (PARTITION BY accid
ORDER BY txdate
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) run_bal
FROM transactions;
```

- https://www.mysqltutorial.org/mysql-window-functions/
- https://www.red-gate.com/simple-talk/databases/sql-server/t-sql-programming-sql-server/window-functions-in-sql/

7. GROUP_CONCAT()

```
SELECT deptno, GROUP_CONCAT(ename) FROM emp
GROUP BY deptno;
```

8. Years in which Employees not recruited,

```
-- years in which emps were recruited
SELECT DISTINCT(YEAR(hire)) yr FROM emp;

SELECT YEAR(hire) yr, COUNT(empno) FROM emp
GROUP BY YEAR(hire);
```

```
WITH RECURSIVE years(yr)
AS (
    SELECT 1975
    UNION ALL
    SELECT yr + 1
    FROM years
    WHERE yr < 1985
)
SELECT * FROM years;

WITH RECURSIVE years(yr)
AS (
    SELECT 1975
    UNION ALL
    SELECT yr + 1
    FROM years
    WHERE yr < 1985
)
SELECT yr FROM years WHERE
yr NOT IN (SELECT DISTINCT(YEAR(hire)) FROM emp);
```

9. Number of records in output with different kinds of join.

```
CREATE TABLE t1 (c1 CHAR(1));
CREATE TABLE t2 (c2 CHAR(1));

INSERT INTO t1 VALUES ('A'), ('B'), ('C'), ('P'), ('Q');
INSERT INTO t2 VALUES ('A'), ('B'), ('B'), ('Y'), ('Q');
```

```sql
-- Inner Join
SELECT t1.c1, t2.c2 FROM t1 INNER JOIN t2 ON t1.c1 = t2.c2;
```

```
+------+------+
| c1   | c2   |
+------+------+
| A    | A    |
| B    | B    |
| B    | B    |
| Q    | Q    |
+------+------+
4 rows in set
```

```sql
-- Left Join
SELECT t1.c1, t2.c2 FROM t1 LEFT OUTER JOIN t2 ON t1.c1 = t2.c2;
```

```
+------+------+
| c1   | c2   |
+------+------+
| A    | A    |
| B    | B    |
| B    | B    |
| C    | NULL |
| P    | NULL |
| Q    | Q    |
+------+------+
6 rows in set
```

```sql
-- Right Join
SELECT t1.c1, t2.c2 FROM t1 RIGHT OUTER JOIN t2 ON t1.c1 = t2.c2;
```

```
+------+------+
| c1   | c2   |
+------+------+
| A    | A    |
| B    | B    |
| B    | B    |
| NULL | Y    |
| Q    | Q    |
+------+------+
5 rows in set
```

```sql
-- Full Outer Join (Oracle)
SELECT t1.c1, t2.c2 FROM t1 FULL OUTER JOIN t2 ON t1.c1 = t2.c2;
```

```
C1  C2
A   A
B   B
B   B
-   Y
Q   Q
C    -
P    -
7 rows in set
```

10. For the below two tables, what will be output of inner join, left join and full outer join?

    - Table

t1	t2
1	1
1	1
200	1
null	400
0	null

11. Explain UNION, UNION ALL, INTERSECT, MINUS.

```sql
(SELECT c1 FROM t1) UNION (SELECT c2 FROM t2);

(SELECT c1 FROM t1) UNION ALL (SELECT c2 FROM t2);

SELECT c1 FROM t1
MINUS
SELECT c2 FROM t2;
-- Oracle: C, P

SELECT c1 FROM t1
LEFT JOIN t2 ON c1 = c2
WHERE t2.c2 IS NULL;
-- MySQL: C, P

SELECT c1 FROM t1
INTERSECT
SELECT c2 FROM t2;
```

```
-- Oracle: A, B, Q

SELECT DISTINCT c1 FROM t1
INNER JOIN t2 ON c1 = c2;
-- MySQL: A, B, Q
```

12. Find records in a table which are not present in another table.

```
(SELECT c1 FROM t1) EXCEPT (SELECT c2 FROM t2);
-- same as MINUS operator (using subquery)
```

13. Find employees with salary more than their manager's salary.

```
SELECT e.ename, e.sal, m.ename, m.sal FROM emp e
RIGHT JOIN emp m ON e.mgr = m.empno
WHERE e.sal > m.sal;
```

14. Update a table and swap gender values.

```
CREATE TABLE people(id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(20), gender CHAR(1));

INSERT INTO people(name, gender)
VALUES
('p1', 'M'),
('p2', 'F'),
('p3', 'M'),
('p4', 'F'),
('p5', 'M');

SELECT * FROM people;
```

```sql
UPDATE people SET gender=CASE
WHEN gender='M' THEN 'F'
ELSE 'M'
END;

SELECT * FROM people;

UPDATE people SET gender=IF(gender='M', 'F', 'M');

SELECT * FROM people;
```

15. DELETE vs TRUNCATE vs DROP
    - DELETE
      - DML operation.
      - Can be rollbacked.
      - Data is removed, but metadata (table structure) is intact.
      - Internals: Typically mark rows as deleted in data files.
    - TRUNCATE
      - DDL operation.
      - Cannot be rollbacked.
      - Data is removed, but metadata (table structure) is intact.
      - Internals: Delete the rows and truncate the data files (to keep only table structure).
    - DROP
      - DDL operation.
      - Cannot be rollbacked.
      - Data as well as metadata (table structure) is deleted.
      - Internals: Whole data files are deleted.

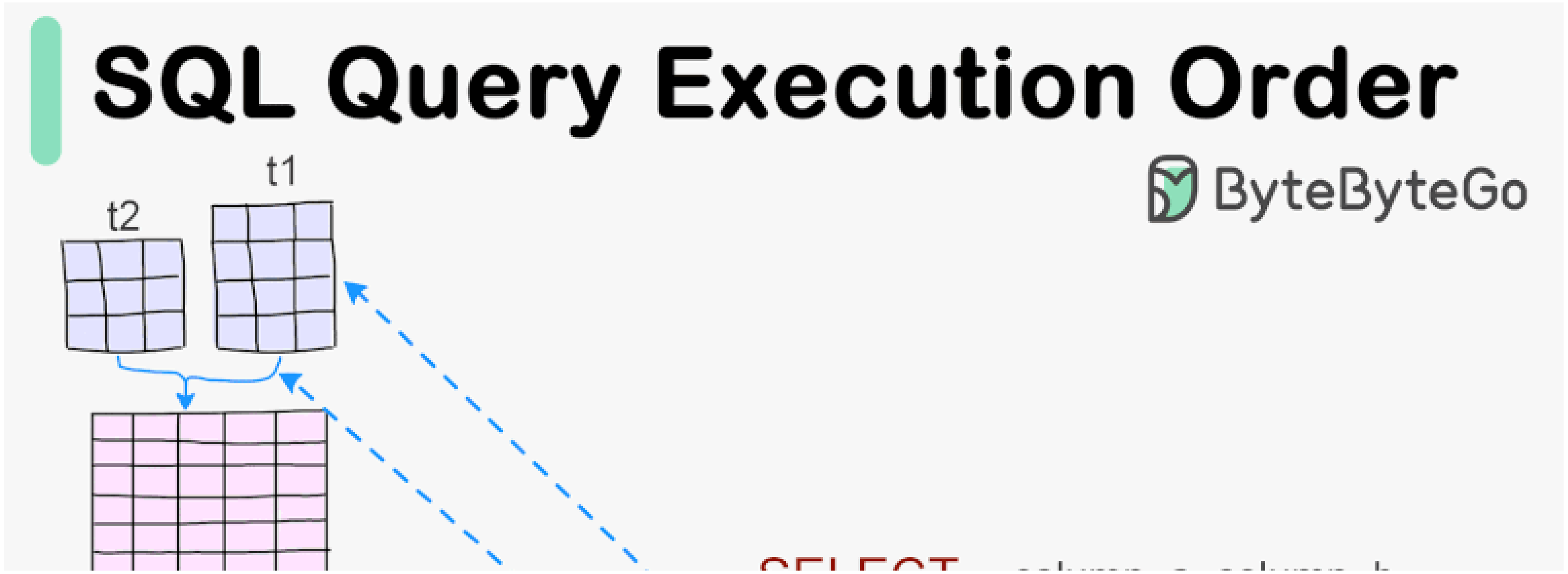
16. Explain Constraints. Unique Constraint vs Primary Key Constraint.

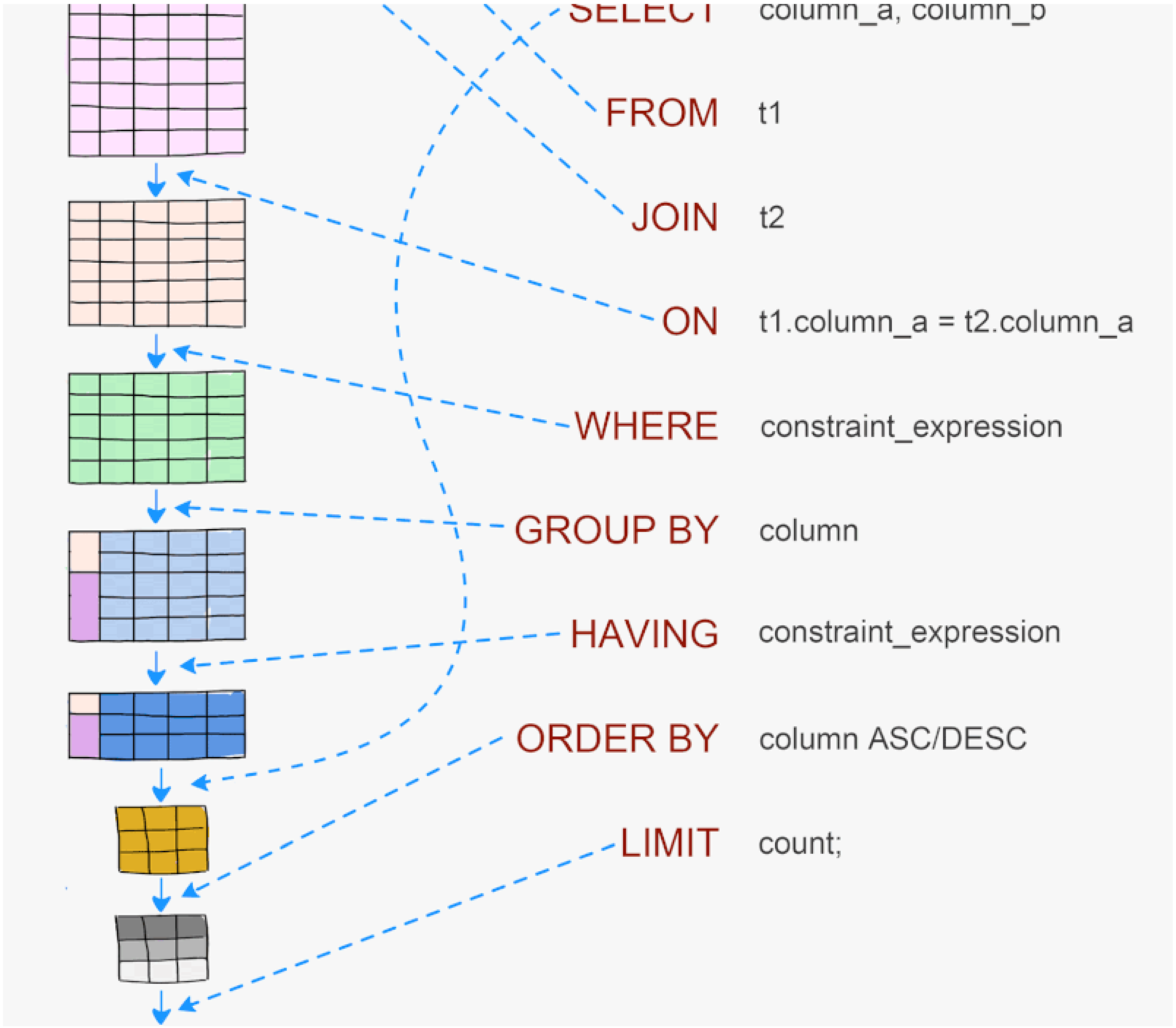
17. At the end how many rows will be in the table?

    - 0 rows initially
    - Insert 2 rows
    - Commit
    - Insert one row
    - RollBack
    - Delete one row
    - Commit
    - Insert two rows
    - Commit
    - Delete all
    - Roll back
    - Truncate
    - RollBack

18. View vs Materialized view. Can name of view and table be same?

19. How SELECT query is executed? Sequence of clauses?




SELECT column_a, column_b
FROM t1
JOIN t2
ON t1.column_a = t2.column_a
WHERE constraint_expression
GROUP BY column
HAVING constraint_expression
ORDER BY column ASC/DESC
LIMIT count;

-

20. Explain normalization and de-normalization.

    - **Normalization **
        - Concept of table design --> Table, Structure, Data Types, Width, Constraints, Relations.
        - Goals:
            - Efficient table structure
            - Avoid data redaundency i.e. unnecessary duplication of data (to save disk space)
            - Reduce problems of insert, update & delete.
        - Done from input perspective.
        - Based on user requirements.
        - Part of software design phase.
        - View entire appln on per transaction basis & then normalize each transaction separately.
    - **Transaction Examples:**
        - Banking: Open New Account, Deposit Amount, Withdraw Amount.
        - Rail Reservation: Reservation, Cancellation.
        - Online Shopping: Customer Order, Stock Update, New Product, ...
    - **Getting ready for Normalisation:**
        - For given transaction make list of all the fields.
        - Strive for atomicity.
        - Get general description of all field properties.
        - For all practical purposes we can have a single table with all the columns. Give meaningful names to the table.
        - Assign datatypes and widths to all columns on the basis of general desc of fields properties.
        - Remove computed columns.
        - Assign primary key to the table.
        - At this stage data is in unnormalised form.

- **UNF --> starting point of normalisation.**
  - Delete anomaly
  - Update anomaly
  - Insert anomaly
- **Normalization steps**
  1. Remove repeating group into a new table.
  2. Key elements will be PK of new table.
  3. (Optional) Add PK of original table to new table to give us Composite PK.
     - Repeat steps 1-3 infinitely -- to remove all repeating groups into new tables.
     - This is 1-NF. No repeating groups present here. One to Many relationship between two tables.
  4. Only table with composite PK to be examined.
  5. Those cols that are not dependent on the entire composite PK, they are to be removed into a new table.
  6. The key eles on which the non-key eles were originally dependent, it is to be added to the new table, and it will be the PK of new table.
     - Repeat steps 4-6 infinitely -- to seperate all non-key eles from all tables with composite primary key.
     - This is 2-NF. Many-to-Many relationship.
  7. Only non-key eles are examined for inter-dependencies.
  8. Inter-depedent cols that are not directly related to PK, they are to be removed into a new table.
  9. (a) Key ele will be PK of new table.
  10. (a) The PK of new table is to be retained in original table for relationship purposes.
      - Repeat steps 7-9 infinitely to examine all non-key eles from all tables and separate them into new table if not dependent on PK.
      - This is 3-NF.
  - To ensure data consistecy (no wrong data entered by end user).

    - Seperate table to be created of well-known data. So that min data will be entered by the end user.
    - This is BCNF or 4-NF.

21. OLAP vs OLTP.

22. How to optimize the query?

    - Optimizing SQL queries is essential for improving the performance of database operations. Here are some strategies you can employ:
        - Use Indexes: Indexes help speed up data retrieval by providing quick access to rows in a table. Analyze your query execution plans and create indexes on columns frequently used in WHERE, JOIN, and ORDER BY clauses.
        - Limit the Data Returned: Only fetch the columns you need and use the LIMIT clause to restrict the number of rows returned, especially when dealing with large datasets. This reduce network traffic between server and client.
        - Avoid SELECT *: Instead of selecting all columns from a table, explicitly specify the columns you need. This reduces the amount of data transferred between the database and the application.
        - Optimize JOINs: Use INNER JOIN, LEFT JOIN, or other types of joins appropriately based on the relationship between tables. Ensure that join conditions are correct (ON clause) and applied on indexed columns.
        - Use EXISTS or IN: Instead of using multiple OR conditions, consider using EXISTS or IN clauses, which can be more efficient, especially when correlated subqueries are involved.
        - Avoid Nested Queries: Nested queries can sometimes be inefficient. Whenever possible, rewrite them as JOINs or EXISTS subqueries.
        - Optimize WHERE Clause: Ensure that the WHERE clause filters rows efficiently by using indexed columns and avoiding unnecessary functions or calculations.
        - Use Stored Procedures: Precompiled stored procedures can improve performance by reducing network traffic and providing better execution plans.
        - Monitor and Analyze Performance: Regularly monitor query performance using database profiling tools. Identify slow queries and optimize them based on execution plans and performance metrics. You can use EXPLAIN statement.
        - Partitioning: For large tables, consider partitioning data based on a key, such as date or region. This can improve query performance by reducing the amount of data that needs to be scanned.
        - Denormalization: In some cases, denormalizing tables by adding redundant data can improve query performance by reducing the need for JOINs and aggregations.

23. Stored Procedure vs Function vs Trigger.

```sql
-- get sal of given emp.
DELIMITER $$

CREATE PROCEDURE sp_getsal(IN pempno INT, OUT psal DOUBLE)
BEGIN
    SELECT sal INTO psal FROM emp WHERE empno=pempno;
END;
$$

DELIMITER ;
```

```sql
-- call sp
CALL sp_getsal(7900, @sal);
SELECT @sal;
```

```sql
DELIMITER $$

CREATE FUNCTION fn_calcincome(psal DOUBLE, pcomm DOUBLE)
RETURNS DOUBLE
DETERMINISTIC
BEGIN
    DECLARE vincome DOUBLE;
    SELECT IFNULL(psal, 0) + IFNULL(pcomm, 0) INTO vincome;
    RETURN vincome;
END;
$$

DELIMITER ;
```

```
SELECT ename, sal, fn_calcincome(sal, comm) income FROM emp;
```

```
CREATE TABLE sal_change(empno INT, oldsal DOUBLE, newsal DOUBLE, modified TIMESTAMP);

DELIMITER $$

CREATE TRIGGER trig_salchange
AFTER UPDATE ON emp FOR EACH ROW
BEGIN
    IF new.sal != old.sal THEN
        INSERT INTO sal_change VALUES (new.empno, old.sal, new.sal, NOW());
    END IF;
END;
$$

DELIMITER ;
```

```
UPDATE emp SET sal=sal+100 WHERE job='CLERK';

SELECT * FROM sal_change;
```