SQL

29/7/25.

- **Database :-** It is a collection of data in a format that can be easily accessed. A software application used to manage our DB is called DBMS.

- **Types :-** (i) **Relational :-** Data stored in tables.
  (ii) **Non-relational :-** Data is not stored in tables (MongoDB)

- **SQL :-** Structured Query language
  It is a programming language used to interact with RDBMS.
  It is used to perform CRUD operations :-
  * Create * Read * Update * Delete.

15/8/25

- **Table :-**
  Columns → structures / schema is defined
  rows → individual data.

- **Syntax :-** CREATE DATABASE name; → to create a database
  DROP DATABASE name; → to delete a database

- **Creating a table :-**

```
CREATE DATABASE student;
USE student;
CREATE TABLE class(
        id INT PRIMARY KEY,
        name VARCHAR(50),
        age INT NOT NULL
    );
INSERT INTO class (1, "ABHI", 22);
INSERT INTO class (2, "LIKI", 20);
SELECT * FROM class;
```

- **Datatypes :-** Signed or unsigned.

__/__/____

- Types of SQL Commands :-
  - * DDL (Data Definition language) : create, alter, rename, truncate & drop.
  - * DQL (Data Query language) : select.
  - * DML (Data Manipulation language) : insert, update & delete.
  - * DCL (Data Control language) : grant & revoke permissions to users.
  - * TCL (Transaction Control language) : start transactions, commit, rollback.

- Database related queries :-
  - * CREATE DATABASE name;
    CREATE DATABASE IF NOT EXISTS name;
  - * DROP DATABASE name;
    DROP DATABASE IF EXISTS name;
  - * SHOW DATABASES;
  - * SHOW TABLES;

- Table related queries :-
  - * Create

```
CREATE TABLE table_name(
 column_name1 datatype constraint;
 );
```

PRIMARY KEY → Not Null, Unique.

  - * Select & View all columns.

SELECT * from table_name;

SELECT col1, col2 FROM table_name; SELECT DISTINCT col FROM name;

  - * Insert

INSERT INTO table_name
(colname1, colname2)

VALUES
(col1 v_1, col2_v1),
(col1_v2, col2_v2);

- Primary key:
It is a column in a table that uniquely identifies each row. There is only 1PK & it should not be NULL.

- Foreign key:
A foreign key is a column in a table that refers to the primary key of other table. There can be multiple FKs. FKs can have duplicate & null values.

- CONSTRAINTS: SQL constraints are used to specify rules for data in a table.

* NOT NULL: columns cannot have null value.
* UNIQUE: all values in column are different.
* PRIMARY KEY: makes a column unique & not null but used only for one.
* FOREIGN KEY: prevents actions that would destroy links between tables.

Eg:- CREATE TABLE temp (
cust_id int, ← foreign key
FOREIGN KEY (cust_id) references customers (id) ← foreign table
);

* DEFAULT: Sets the default value of a column.
* CHECK: It can limit the values allowed in a column.

Eg:- CREATE TABLE newtab (
age INT CHECK (age >= 18)
);

16/8/25

- SELECT in detail :-
  - * To view all : SELECT * FROM table_name;
  - * Basic : SELECT col1, col2 FROM table_name;
  - * Unique value : SELECT DISTINCT col FROM table_name;
  - * Where clause : SELECT col, col2 FROM table_name WHERE conditions;

- Operators used in while clause :-
  - * Arithmetic operators : +, −, *, %, /.
  - * Comparison operators : =, !=, >=, <=, >, <.
  - * Logical Operators : AND, OR, NOT, IN, BETWEEN, ALL, LIKE, ANY.
  - * Bitwise operators : &, |.

- * LIMIT clause :
  Sets an upper limit on number of rows to be returned
  SELECT * FROM table_name
  LIMIT number;

- Order By clause :
  To sort in ascending (ASC) or descending (DESC) order
  SELECT col1, col2 FROM table_name
  ORDER BY col_name (s) ASC;

- AGGREGATE FUNCTIONS :-
  Aggregate functions perform a calculation on a set of values, & return a single value.
  - * COUNT()
  - * SUM()
  - * MAX()
  - * AVG().
  - * MIN()

- GROUP BY clause :
  Groups rows that have the same values into summary rows.
  It collects the data from multiple records & groups the result

by one or more column.
Generally we use group by with some aggregate function.
SELECT city, count(name) FROM student GROUP BY city;

Eg:
```
CREATE DATABASE temp9;
USE temp9;
CREATE TABLE student(
 rollno INT PRIMARY KEY,
 name VARCHAR(50),
 city VARCHAR(10),
 marks INT NOT NULL);
INSERT INTO student
(ROLLNO, name, city, marks)
VALUES
(60, "Gojo satoru", "Shibuya", 100),
(61, "Sung Jinwoo", "Seoul", 90),
(62, "Kaizehaya", "Kyoto", 67),
(63, "Tanjiro", "Kobe", 85);
SELECT NAMES, avg(marks)
FROM student
GROUP BY NAME
ORDER BY avg(MARKS) DESC;
```

- For the given table, find the total payment according to each payment method

```
CREATE DATABASE japan;
USE japan;
CREATE TABLE company (
    customer_id INT PRIMARY KEY,
    customer VARCHAR(40),
    mode VARCHAR(50),
    city VARCHAR(20)
);
```

```
INSERT INTO company (customer_id, customer, mode, city)
VALUES
(101, "Gojo Satoru", "Netbanking", "Shibuya"),
(102, "Sung Jinwoo", "Credit Card", "Seoul"),
(103, "Izuku Midoriya", "Credit Card", "Kobe"),
(104, "Zenitsu", "NetBanking", "Osaka"),
(105, "Kazehaya", "Netbanking", "Tokyo"),
(106, "Tanjiro", "Debit Card", "Kyoto"),
(107, "Okarun", "Netbanking", "Saitama"),
(108, "Miyamura", "Debit Card", "Shinjuku");
SELECT mode, count (customer)
FROM company
GROUP BY mode;
```

Op :-

mode	count (customer)
Netbanking	4
Credit card	2
Debit Card	2

- **HAVING Clause** : Applies some conditions on groups.
  Used when we want to apply any condition after grouping.

Eg :-
```
SELECT city, count (rollno)
FROM student
GROUP BY city
HAVING MAX(marks) > 90;
```

- **General order** :
```
SELECT column(s)
FROM table_name
WHERE condition
GROUP BY column(s)
HAVING condition
ORDER BY column(s) ASC;
```

__/__/____

- UPDATE :- to update existing rows.

```
UPDATE table_name
SET col1 = val1, col2 = val2
WHERE condition;
```

- Safe update error : SET SQL_SAFE_UPDATES=0;

- DELETE : to delete existing rows.

```
DELETE FROM table_name
WHERE condition;
```

18/8/25

- Cascading for FK :

ON DELETE CASCADE

When we create a foreign key using this opinion, it deletes the referencing rows in the child table when the referenced row is ~~deleted~~ deleted in the parent table which has a primary key.

ON UPDATE CASCADE

When we create a foreign key using UPDATE CASCADE the referencing rows are updated in the child table when the reference row is updated in the parent table which has a primary key.

Eg :-

```
CREATE TABLE student (
    id INT PRIMARY KEY,
    courseID INT,
    FOREIGN KEY (courseID) REFERENCES course (id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

18/8/25

→

Eg :- CREATE TABLE dept(
id INT PRIMARY KEY,
dept_name VARCHAR(50)
);

→ INSERT INTO dept (id, dept_name)
VALUES
(123, 'Science')
(897, 'Mathematics')

→ CREATE TABLE teachers (
id INT PRIMARY KEY,
name VARCHAR(50),
dept_id INT,
FOREIGN KEY (dept_id) REFERENCES dept(id)
ON UPDATE CASCADE
ON DELETE CASCADE
);

→ INSERT INTO teachers
VALUES
(10, 'MINAL', 123),
(29, 'SOPHIA', 897);
SELECT * FROM teachers;
SELECT * FROM dept;

→ UPDATE dept
SET id = 129
WHERE id = 123;

⇒ Table related queries :-

- ADD column
ALTER TABLE table_name
ADD COLUMN column_name datatype constraint;

- Drop column

ALTER TABLE table_name
DROP COLUMN column_name;

- RENAME table
ALTER TABLE table_name
~~DROP COLUMN column_name;~~ RENAME TO new_table_name;

- CHANGE Column(rename)
ALTER TABLE table_name
CHANGE COLUMN old_name new_name new_datatype
new_constraint;

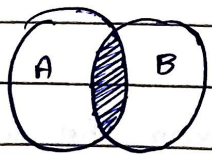
- MODIFY Column (modify datatype/constraint)
ALTER TABLE table_name
MODIFY col_name new_datatype new_constraint;

- TRUNCATE TABLE table_name;
To delete table's data.

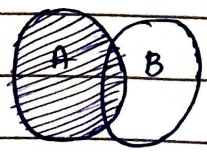
→ Joins in SQL : Join is used to combine rows from two or more tables, based on related column between them.

→ Types of Joins :

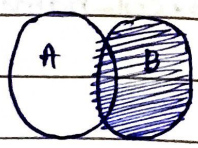
- Inner Join :



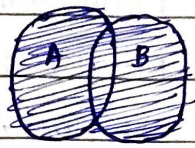
- Outer Join :
  - Left join :



  - Right join :



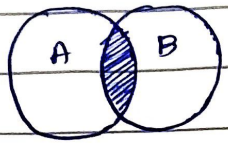
  - Full Join :



__/__/____

- **Inner Join :**

Returns records that have matching values in both sides.



Syntax : SELECT column(s)
FROM tableA
INNER JOIN tableB
ON tableA.col_name = tableB.col_name;

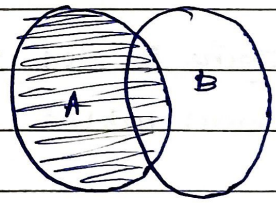
- **alias (alternate name) :**

Eg :- You have 2 ~~colu~~ tables named anime & manga.
∴ I am performing inner join here for simplicity,

```
SELECT *
FROM anime as a
INNER JOIN manga as m
ON a.id = m.id.
```

- **Left Join :**

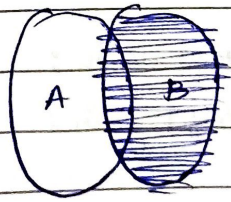
Returns all records from the left table, & the matched records from the right table.



Syntax : SELECT column(s)
FROM tableA
LEFT JOIN tableB
ON tableA.col_name = tableB.col_name;

- **Right Join :**

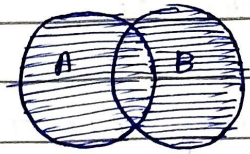
Returns all records from the right table, & the matched records from the left table.



Syntax : SELECT column(s)
FROM tableA
RIGHT JOIN tableB
ON tableA.col_name = tableB.col_name;

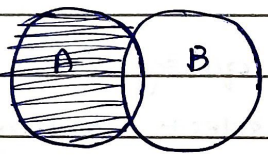
- **Full Join:**

Returns all records when there is a match in either left or right table.



Syntax:
```
SELECT * FROM anime as a
LEFT JOIN manga as m
ON a.id = m.id
UNION
SELECT * FROM anime as a
RIGHT JOIN manga as m
ON a.id = m.id;
```

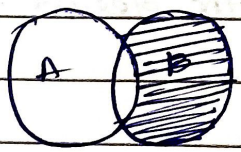
- **Left Exclusive Join:**



Syntax:
```
SELECT *
FROM anime as a
LEFT JOIN manga as m
ON a.id = m.id
WHERE m.id IS NULL;
```

Returns only the record which is not common to both tables & doesnot return the other tables data.

- **Right Exclusive Join:**



Syntax:
```
SELECT *
FROM anime as a
RIGHT JOIN manga as m
ON a.my_id = m.id
WHERE a.my_id IS NULL;
```

- **Self Join:**

It is a regular join but the table is joined with itself.

Syntax:
```
SELECT *
FROM anime as a
JOIN manga as m
ON a.id = m.id;
```

__/__/____

- UNION :

It is used to combine the result-set of two or more Select statements.

→ To use it :

* every SELECT should have same no. of columns
* columns must have similar data types.
* columns in every SELECT should be in same order.

Syntax : SELECT column(s) FROM table A
UNION
SELECT column(s) FROM table B.

union all : gives duplicate names too.

→ SQL Sub Queries :-

A subquery or inner query or nested query is a query within another SQL query. It involves 2 select statements.

Syntax : SELECT column(s)
FROM table_name
WHERE col_name operator
(subquery);

→ My SQL views :- (Virtual data).

A view is a virtual table based on the result-set of an SQL statement.

Syntax : CREATE VIEW view1 AS
SELECT rollno, name FROM student;
SELECT * FROM view1;

① Eg : SELECT AVG(marks)
FROM student;

SELECT name, marks
FROM student
WHERE marks > (SELECT AVG(marks) FROM student);

(2). SELECT name, id
FROM anime_scores
WHERE id IN (SELECT id FROM anime_scores WHERE id % 2 = 0);

- Eg with FROM.
SELECT MAX (marks)
FROM (SELECT * FROM anime_scores WHERE city = "Tokyo")
AS ttt;

- Eg with SELECT.
SELECT (SELECT MAX(marks) FROM anime_scores), name
FROM anime_scores;

- A view always shows up-to-date data. The database engine recreates the view, every time a user queries it.