DATE: / /

PAGE: _____

SQL Notes :-

SQL :- is a Programming Language used to interact with db.

Create
Read
Update
Delete

Relational Database	Non-Relational ~~DBA~~ DB
1) SQL Database	NoSQL database
2) Data Stored in tables	Data stored in either key-value pairs, document-based, graph database or wide column stores
3) dbs have fixed or static or predefined schemas	They have dynamic schemas
4) Less performance with huge volumes of data Eg :- PostgreSQL, MYSQL, MS SQL Server	Easily scale with huge volumes of data E.g MongoDB, Cassandra, Hbase

DATE: / /
PAGE: _____

Types of SQL Commands:

DDL (Data definition Language)	DML (Data manipulation language)	DCL (Data Control Language)
- Changes in schema	- Changes in data	- Changes in access authorities
- Create - alter - DROP - TRUNCATE	- INSERT - UPDATE - DELETE	- GRANT - REVOKE

Database: System that allows users to store and organise data

Excel	Database
1) Easy to use, for trained can work	trained person required
2) data stored less data	stores large amount of data
3) Good for one time analysis, quick charts	Can automate tasks
4) No data Integrity due to manual operation.	High data Integrity
5) Low search/filter capabilities	High search/filter capabilities

DATE: / /
PAGE: _____

→ datatypes :-

int	varchar
float	date
bool	datetime
char	

★ Keys :-

1) Primary key (PK) :-
→ unique column we set in table to easily identify and locate data in queries
⇒ unique & not NULL

2) Foreign key (FK) :-
→ column used to link b/w two or more tables together
→ any number of FK can have duplicate and NULL values

★ Constraints

→ used to specify rules for data in table
→ ensure accuracy & reliability of data
→ specified when table is created or with alter table

e.g CREATE TABLE table name (
column1 datatype constraint;
...
);

DATE: / /
PAGE:

→ NOT NULL
→ UNIQUE
→ PRIMARY KEY
→ FOREIGN KEY → prevents actions that would destroy links b/w tables
→ CHECK → value ~~specify~~ satisfies a specific condition.
→ DEFAULT → sets default value for column when no value specified.
→ CREATE INDEX → used to create & retrieve data from database very quickly.

→ INSERT INTO table_name
(col_1, col_2, col_3, ...)
VALUES
(value1, value2, value3, ...);

→ UPDATE tablename
SET col_1 = val1, col_2 = val2 ...
WHERE ID = 'value';

→ DELETE FROM tablename
WHERE condition;

→ ALTER TABLE tablename
ADD COLUMN columnname;

→ ALTER TABLE table_name
DROP COLUMN column_name;

→ ALTER TABLE table_name
MODIFY ALTER COLUMN column_name;

→ DROP TABLE table_name;

→ TRUNCATE TABLE table_name;

→ SELECT * FROM table_name;

→ SELECT DISTICT Column_name FROM table_name;

OPERATORS IN SQL :-

1) +, -, *, /, % ⇒ ARITHMETIC
2) =, !=, >, >= ⇒ Comparison
3) ALL, IN, BETWEEN, LIKE, AND, OR, NOT, ANY
4) &, | ⇒ Bitwise

→ LIMIT :-

SELECT column_name FROM table_name
LIMIT 10;

→ ORDER BY :-

SELECT column_name FROM table_name
ORDER BY colum_name ASC / DESC,

DATE: / /
PAGE: _____

→ IMPORT DATA FROM CSV TO DB:

COPY customer (same columns as csv)
FROM 'path'
DELIMITER ','
CSV HEADER;

→ FUNCTIONS :-

round() sum()
round() max()
lower() MIN()
count()

→ STRING FUNCTION :-

UPPER() ⇒ convert value in uppercase
LOWER() ⇒ lowercase
~~LENGTH~~ LENGTH() ⇒ length
SUBSTRING() ⇒ substring finding
NOW() ⇒ current system date & time
FORMAT() ⇒ used to set format of field
CONCAT() ⇒ add two or more
REPLACE() ⇒ Replace all occurance with new string
TRIM() ⇒ remove leading & trailing white spaces from string

SUBSTRING(col_name, 1, 3)
                start, len
REPLACE(col_name, 'MARY', 'MOHAN')
                ↑ Replaced ↑ with this

DATE: / /
PAGE: _____

→ AGGREGATE FUNCTION:

Paforms actions with multiple ~~statements~~ values & return single value.

often used with GROUP BY & SELECT statement

COUNT() → return number of values (How many entry)

SUM() → return sum of all values

AVG() ⇒ returns average value

MAX() ⇒ " maximum value

MIN() ⇒ return minimum value

ROUND() ⇒ " specified number of decimal places

ROUND(value, noofdigits)

→ GROUP BY group rows that have same value into summary rows.

→ often used with aggregate functions

```
SELECT column_name
FROM table_name
GROUP BY column_name;
```

DATE: / /
PAGE: _____

eg
```
SELECT mode, SUM(amount) AS total
FROM payment
GROUP BY mode
ORDER BY total ASC
```

★ HAVING Clause

→ used to apply filter on result of GROUP BY based on specified condition.

→ it is same as where in select.

```
SELECT      column_name
FROM        table_name
WHERE       condition(s)
GROUP BY    column_name
HAVING      condion(s)
```

e.g
```
SELECT mode, COUNT(amount)
AS total
FROM payment
GROUP BY mode
HAVING COUNT(amount) >= 3
ORDER BY total DESC
```

★ TIME STAMP

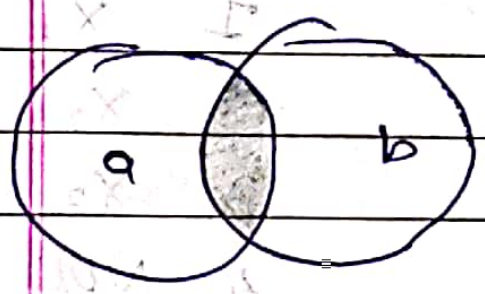
TIME : HH:MI:SS
DATE : YYYY-MM-DD

DATE: \ \
PAGE: _____

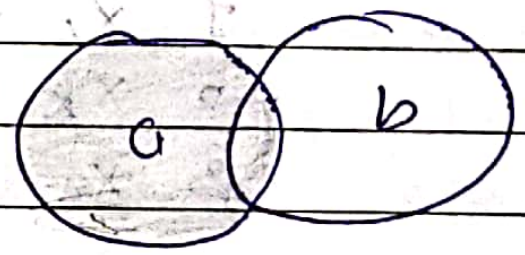
YEAR : YYYY
TIMESTAMP : date & time YYYY-MM-DD HH:MI:SS
TIMESTAMPtz : date, time, time zone

★ JOINS :-

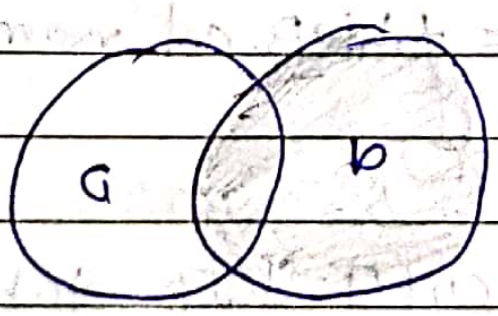
- Types of Join :
  1) INNER JOIN
  2) LEFT JOIN
  3) RIGHT JOIN
  4) FULL JOIN



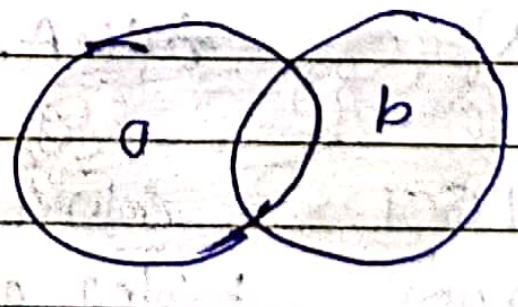
a INNER JOIN b



a LEFT JOIN b



a RIGHT JOIN b



a FULL JOIN b

SELECT column_name(s)
FROM ~~table_name~~ A INNER JOIN B
ON A.col_name = B.col_name

DATE: / /
PAGE: _____

```
SELECT *
FROM Customer AS C INNER JOIN
     payment AS p
ON   C.customer_id = p.customer_id
```

LEFT JOIN :-

INNER JOIN ⇒

1	x_1		1	y_1	
2	x_2		2	y_2	
3	x_3		4	y_4	

RIGHT JOIN ↓ | ↓ LEFT JOIN | FULL JOIN ↓

1	x_1	y_1	1	x_1	y_1	1	x_1	y_1
2	x_2	y_2	2	x_2	y_2	2	x_2	y_2
4	NULL	y_4	3	x_3	NULL	3	x_3	NULL
						4	NULL	y_4

→ SELECT column_name(s)
FROM tableA LEFT JOIN tableB
ON tableA.col_name = tableB.col_name

→ SELECT column_name(s)
FROM table1 as A RIGHT JOIN table2 as B
ON A.col_name = B.col_name

→ SELECT column_name(s)
FROM table1 as A FULL OUTER JOIN
table2 as B
ON A.col_name = B.col_name

DATE: \ \
PAGE: _____

INNER Join : maching values in both tables
LEFT JOIN : when required data is present in left table
RIGHT JOIN : where " " " " right "
FULL OUTER JOIN : either left or right / all data

→ SELF JOIN :-

while comparing different columns in a same table

```
SELECT column-name(s)
FROM   Table As T1
JOIN   Table As T2
ON     T1.col_name = T2.colname
```

Emp_id	emp_name	manager_id
1	Agni	3
2	Akash	4
3	Dharti	2
4	Vayu	3

// selfjoin

emp_id	emp_name	manager_id	mana_name
1	Agni	3	Dharti
2	Akash	4	Vayu
3	Dharti	2	Akash
4	Vayu	3	Dharti

```
SELECT *
FROM   emp As T1
JOIN   emp As T2
WHERE ON   T1.emp-id = T2.manager-id
```

DATE: / /
PAGE:

★ UNION :-

→ combine / concatenate results without duplicate rows and keeps unique records

- Same number of columns selected and expressions
- Same data type
- Same order of columns.

```
SELECT column_name(s) FROM TableA.
UNION
SELECT column_name(s) FROM TableB.
```

If duplicates required then

```
SELECT column_name FROM TableA
UNION ALL
SELECT column_name FROM TableB;
```

★ SUBQUERY :- / INNERQUERY / NESTED QUERY

e.g. : display salary of employee which is greater than avg. salary of employees:

```
SELECT salary FROM payment
WHERE salary > (SELECT AVG(salary)
FROM payment);
WHERE
```

DATE: / /
PAGE: _____

When comparison is having more than one value

```
SELECT customer_id, amount, mode
FROM Customer
WHERE customer_id IN (SELECT customerid
                      FROM customer)
```

// Alternative of Join

```
SELECT first_name, last_name
FROM custome c
WHERE EXISTS ( SELECT customerid, amount
               FROM payment p
               WHERE p.customer_id = c.customerid
               AND amount > 100)
```

WINDOW FUNCTION :-

→ applies aggregate, ranking, analytic function over particular window

→ OVER Clause is need

WINDOW FUNCTIONS :-

- AGGREGATE
  - SUM
  - AVG
  - MIN
  - MAX
  - COUNT
- RANKING
  - ROW_NUMBER
  - RANK
  - DENSE_RANK
  - PERCENT_RANK
- ANALYTIC VALUE
  - LEAD
  - LAG
  - FIRST_VALUE
  - LAST_VALUE

row number → give a row number

don't skip

DATE: \ \
PAGE: _____

```
SELECT new_id, newcat,
   SUM (new_id) OVER (PARTITION BY newcat
          ORDER BY new_id) AS "Total",
   COUNT(new_id) OVER (PARTITION BY newcat
          ORDER BY new_id) AS "Count",
FROM test_data
```

→ CASE statement:

→ it similar to if-else condition
→ it No else part or no condition true it returns NULL:

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
END;
```

e.g

```
SELECT customer_id, amount,
CASE
    WHEN amount > 100 THEN
        'Expensive product'
    WHEN amount = 100 THEN
        'Moderate product'
    ELSE 'Inexpensive product'
END AS ProductStatus
FROM payment
```

DATE: \ \
PAGE: _____

* COMMON TABLE EXPRESSION (CTE) :-

→ temporary result set created from simple SELECT statement which can be used in subsequent SELECT statement.

```
WITH my_cte AS (
        SELECT a,b,c
        FROM Table1)
SELECT a,c
FROM my_cte;
```

e.g

temporary view

```
WITH my_cte AS (
    SELECT *, AVG(amount) OVER(ORDER BY
    P.customer_id) AS "Average_Price",
    FROM payment as P INNER JOIN
            customer as C
        ON  p.customer_id = c.customer_id
    )

SELECT first_name, last_name
FROM my_cte;
```