SQL (Case Insensitive)

SDS Page No
Date / /

① **SQL is a :-**

→ language used to insert, delete or modify the data of relational databases (RDMS)

RDBMS

→ data is stored in form of tables with rows & columns.

Table ⇒ Relation

Row ⇒ Tuple/Record | Column ⇒ Attribute / field

Normalization

– It is a process of organizing data to avoid duplication & redundancy.

Normalization Types

– Normalization Rules are divided into following normal forms

1) First Normal form

– Each set of column must have unique value
– Each row should have a primary key that identifies it uniquely.

ex.

Student	Age	Course
Rosh	22	CR1, CR2
Khush	20	CR2
Deep	18	CR2

Student	Age	Course
Rosh	22	CR1
Rosh	22	CR2
Khush	20	CR2
Deep	18	CR2

1NF

(Data Redundancy / Repition Increase)

Scanned with CamScanner

2) Second Normal form.

- It must satisfy 1NF first along with below condn.
- There must not be any partial dependency of any column on primary key.
- A table that has concatenated primary key, each col in table that is not a part of primary key must depend upon the entire concatenated key for its existence
- If any column depends only on one part of concatenated primary key then table fails 2NF normal form.

ex.

Student	Age	Course
Rosh	22	CR1
Rosh	22	CR2
Khush	20	CR2
Deep	18	CR2

Candidate key {Student, Course}
↑
Concatenated primary key

Here col age that is not a part of Candidate key depends only upon student.
Thus, split table to satisfy 2NF

Student	Age
Rosh	22
Rosh	22
Khush	20
Deep	18

Student	Course
Rosh	CR1
Rosh	CR2
Khush	CR2
Deep	CR2

— Now both satisfies 2NF
— They will never suffer update anomalies

prime attl → cols in candidate key
non prime attl → cols not in candidate key

③ Third Normal Form.

- It must satisfy 1NF & 2NF along with below
- every non prime attribute of table must be dependent on primary key

ex. Student Detail Table.

stu_id	stu_name	DOB	street	city	state	zip

- Here stu_id is primary key, thus all other att must be dependent on student.
- But street city state are dependent on zip
- Thus to satisfy 3NF, move street, city, state to new table with zip as primary key.

i.e Student Detail Table

stu_id	stu_name	DOB	zip

Address Table

zip	street	city	state

Scanned with CamScanner

Datatypes in SQL

- Data type is an attribute that specifies type of data that the object can hold.

ex:
integer data, character date, binary string etc

① character string data type

Datatype	Description
char(n)	stores n characters
nchar(n)	Stores n unicode character
varchar(n)	stores approximately n characters
varchar(max)	Stores upto 2^31-1 characters.
nvarchar(n)	Stores approximately n characters
nvarchar(max)	stores upto ((2^31-1)/2)-2 characters.

② Numeric Datatype

int	stores integer values
tinyint	
smallint	
bigint	
money	
Small money	
decimal(p,s)	stores decimal values of precision p & scales
numeric(p,s)	
float(n)	
real	

③ Date and Time Data Type

date
datetime
datetime2
datetimeoffset
smalldatetime
time

④ Binary Data Type

bit	Stores single bit of data.
binary(n)	Stores n bytes of binary data
varbinary(n)	stores app n bytes of bin data
varbinary(max)	Stores upto 2^31-1 bytes of bin data

Scanned with CamScanner

Basic SQL Statements

① Data Definition language

- Deals with database schemas & Descriptions of how data should reside in Database.
  - Create • Alter • Drop • TRUNCATE

② Data manipulation language

- Deals with Data manipulation & used to store, modify, retrieve, delete & update data in database
  - select • Insert • Update • Delete

③ Data Control language

- Concerned with rights, permissions & other controls of database System
  - Grant ° Revoke

④ Data Transaction Language

- Deals with transaction within a database
  - Commit
  - Rollback
  - Savepoint

SQL Commands

DDL Commands

① Create (DDL)

a) Create Database Object

Syntax : CREATE DATABASE <database_name>

ex : CREATE DATABASE Employees

b) Create Table

Syntax : CREATE TABLE <table_name> (
Column A datatype ,
columnB datatype ,
);

ex : CREATE TABLE emp (
emp_id INT PRIMARY KEY ,
last_name VARCHAR(50) NOT NULL ;
First_name VARCHAR (50) NOT NULL ;
);

Column cannot have null value.

② Alter Table (DDL)

- Used to Add, Delete, modify columns from Existing Table
- Add / Drop constraints on Existing Table.

Scanned with CamScanner

④ Alter Table

Syntax : ① ALTER TABLE <table_name>
ADD <column_name> <datatype>

add column

ex : ALTER TABLE emp ADD Age INT;

Syntax : ② ALTER TABLE <table_name>
DROP column <column_name>

drop column

ex : ALTER TABLE emp DROP Column Age

Syntax : ③ ALTER TABLE <table_name>
ALTER COLUMN <column_name>
<datatype>

alter column constraint

ex : ALTER TABLE emp ALTER COLUMN
first_name varchar (30) ;

⑤ DROP TABLE (DDL)

- used to drop Table from selected Database

Syntax : DROP TABLE <table_name>

ex : DROP TABLE emp ;

⑥ TRUNCATE TABLE (DDL)

- used to delete all rows from table & free the space containing the Table

Syntax → TRUNCATE TABLE <tablename>

* Constraints

- Constraints enforce rules on table whenever rows are inserted, updated & deleted from table
- prevents the deletion of a table if there are dependencies from other Tables
- Define constraints at column level or table level
- Constraints can be applied during creation of table or after creation by using alter command

Constraint	Description
⇒ NOT NULL	Specifies that a column must have some value (it cannot be null)
⇒ UNIQUE	Specifies that column must have unique values (no two rows can have same val for a clmn) & allows null value
⇒ PRIMARY KEY	Specifies a col or set of col that uniquely identifies a row. It doesnot allow null values
⇒ FOREIGN KEY	Foreign key is a column(s) that references a column(s) of a table
⇒ CHECK	Specifies a condn that must be satisfied by all rows in a table
⇒ DEFAULT	sets a default value for a col if no val is specified
⇒ CREATE INDEX	used to create & retrieve data very quickly

Scanned with CamScanner

→ constraint : Primary key [During creation of Table] :

ex. Approach ①

- during creation of
- In this approach, we cannot specify more than 1 col to be primary key (candidate key)

```
CREATE TABLE emp1
(
    emp_id INT PRIMARY KEY,
    l_name VARCHAR(50) NOT NULL
);
```

Approach ②

- after creation
- can specify / include more than 1 col in primary key

```
CREATE TABLE emp2
(
    emp_id INT,
    l_name VARCHAR(50) NOT NULL,
    CONSTRAINT emp_pk PRIMARY KEY (emp_id, l_name).
);
```

(emp_pk ↓ constraint name)

Constraint : Primary key [After creation of Table]

ex

```
CREATE TABLE emp3
(
    emp_id INT NOT NULL,
    l_name VARCHAR(50) NOT NULL,
);
```

add

```
ALTER TABLE emp3 ADD CONSTRAINT emp3_pk
PRIMARY KEY (emp_id) ;
```

drop

```
ALTER TABLE emp3 DROP CONSTRAINT emp3_pk
```

~~Constraint~~

Scanned with CamScanner

Constraint : Foreign key
(During creation of Table)

ex
```
CREATE TABLE Product
(
    prod_id INT PRIMARY KEY,
    prod_name VARCHAR(50) NOT NULL,
);

CREATE TABLE Orders
(
    order_id INT PRIMARY KEY,
    prod_id INT NOT NULL,
    quantity INT,

    CONSTRAINT fk_prod_id FOREIGN KEY(prod_id)
    REFERENCES product(prod_id)
);
```

Referenced Table

Referencing Table

Constraint : Foreign key
After creation of Table using ALTER

ex
```
CREATE TABLE product1
(
    prod_id INT PRIMARY KEY,
    prod_name VARCHAR(50) NOT NULL
);

CREATE TABLE orders1
(
    order_id INT PRIMARY KEY,
    prod_id INT NOT NULL,
    quantity INT
);
```

- ALTER TABLE orders1
  ADD CONSTRAINT fk_prod_id
  FOREIGN KEY (prod_id)
  REFERENCES product1 (prod_id) ;
  ON DELETE CASCADE ;

add

If any val / row of referenced relation gets deleted, then the ~~att~~ referencing att's value will also get deleted

- ALTER TABLE ~~orders1~~ orders1 DROP CONSTRAINT fk_prod_id ;

drop

Scanned with CamScanner

VIEWS

- A view is a named, derived, virtual table
- A view takes o/p of a query & treats it as a table.

① Syntax :

⇒ CREATE VIEW <view_name> AS SELECT * FROM <table_name>;

ex. CREATE VIEW VW_EMP AS SELECT * FROM emp;

To query above view,

SELECT * FROM VW_EMP

CREATE VIEW with Specific columns.

Syntax :

⇒ CREATE VIEW VW_EMP AS
SELECT emp_id,
f_name,
l_name,
job_id
FROM emp.

DML Commands

① INSERT

- used to insert data / record into database table
- inserting values for specific columns in table (always include the columns which are not null)

Syntax :

⇒ INSERT INTO <Table name>
(fieldname1, fieldname2, ...)
VALUES (val1, val2, ...) ;

ex INSERT INTO dept (dept_no, dept_name)
VALUES (50, 'HR') ;

- If you want to insert values in all columns then no need to specify column names, but order of column values should be in sync with column names

ex INSERT INTO dept VALUES (80, 'marketing', 'mumbai') ;

Scanned with CamScanner

INSERT - AS - SELECT

— If you want to insert all records of old table into new table with / without condition

Syntax :

ex:1) INSERT INTO new_dept (Deptno, Dept_name)
(SELECT * from dept)

ex 2) INSERT INTO new_dept (Dept_no, Dept_name)
(SELECT * FROM dept
WHERE Dept_no > 20) ;

i.e

Syntax

⇒ INSERT INTO Table_name (Column names)
SELECT & column names
FROM table_name
WHERE condition

② UPDATE

— update statement updates / modify existing data in table.
— used for :
⇒ updating single column or multiple columns

Syntax :

⇒ UPDATE <table_name> SET <field_name> = <value> WHERE <condition> ;

note ° without WHERE clause, all rows get updated
° while updating multiple columns, column must be separated using ; operator

ex :
UPDATE dept
SET dept_name = 'Computer' ,
dept_loc = 'mumbai'
WHERE dept_no = 20 ;

before

dept_no	dept_name	dept_loc
30	IT	Bangalore
20	HR	Pune

dept_no	dept_name	dept_loc
30	IT	Bangalore
20	Computer	Mumbai

dept_no	dept_name	dept_loc
30	Computer	mumbai
30	Computer	mumbai

without where clause all rows updated

Scanned with CamScanner

③ DELETE

- Delete commands helps to delete rows/ records from database table.
- It can be executed with / without WHERE clause
- execution of delete commands without where condn will delete all rows/ records from table

Syntax :-

⇒ DELETE FROM <table_name> [WHERE <condition>] ;

ex :-

a) deleting without WHERE clause to delete all rows of table.

```
DELETE FROM dept ;
```

b) deleting with WHERE clause to perform conditional deletion of particular row(s) only.

```
DELETE FROM dept
WHERE loc = 'mumbai' ;
```

④ SELECT

- select statement helps us to retrieve records from data table
- where condn is optional in select statements

Syntax :

⇒ SELECT <field1, field2, ...> FROM <tablename> [WHERE <condition>] ;

ex :-

① SELECT * FROM dept; -- fetches all columns in output (with all rows/ records as WHERE clause is not used)

② SELECT top(5) * FROM emp; -- fetches all columns in output with top 5 records

③ SELECT fname, lname FROM emp; --fetches cols fname & lname of Emp

④ SELECT top(5) fname, lname FROM emp; --fetches fname & lname cols with top 5 rows/ records

Scanned with CamScanner

Select statement : Alias name for a field

ex.

- SELECT dept_name, loc from dept;

o/p →

dept_name	loc
Comp	Pune
IT	Binar
HR	chennai

— Using Alias name for a field / column.

— Syntax :-

⇒ SELECT <col1> AS <alias_name1>,
<col2> AS <alias_name2>
FROM <Table_name> ;

ex :-

- SELECT dept_name, loc AS location from dept;

dept_name	location
Comp	Pune
IT	Binar
HR	chennai

— Thus we use alias to improve readability of field / cols.

Select Statement : Distinct values.

— Distinct values :

- used to retrieve unique values for a column.
- multiple rows can have same value for a column, distinct keyword in select statement helps to retrieve unique rows for a column.

Syntax :-

⇒ SELECT DISTINCT < col > from < table_name >

ex :-

dept_id	dept_name	loc
1	IT	Pune
2	HR	mumbai
3	Comp	Pune

SELECT DISTINCT loc FROM dept ;
~~ORDER BY dept_no DESC ;~~

op ⇒

dept_id	dept_name	loc
1	IT	Pune
2	HR	mumbai

Scanned with CamScanner

— Sorting —

ORDER BY

- used along with WHERE clause to display specified col. in ascending or descending order
- default is ascending order.
- Syntax :

⇒ SELECT [distinct] <column(s)>
FROM <table>
[WHERE <condition>]
[ORDER BY <column(s) [asc | desc]>

- ex :-

```
SELECT f_name, dept_no, hiredate
FROM emp
ORDER BY deptno ASC, hire_date desc;
```

① ②

-- fetch all those columns records of dept & order records in descending order : deptno

```
SELECT * FROM dept ORDER BY dept_no DESC;
```

-- fetches bottom 2 records

```
SELECT TOP(2) * from Dept ORDER BY ;
dept_no DESC
```

-- fetches top 2 records

```
SELECT TOP(2) * FROM dept ;
```

— Filtering — logical Operators.

- for filtering we use **WHERE clause**
- **logical Operators (AND , OR and NOT)**
  - used in WHERE conditions to join ~~2 or~~ more than 2 queries.
  - used to combine results of 2 or more conditions to produce single result.

① **AND logical Operator.**

- used to combine 2 conditions & it fetches the result which satisfy both conditions

- ex : SELECT fname, lname FROM emp WHERE fname = 'miller' AND lname = 'WARD'

op

fname	lname
miller	ward

- ex : SELECT fname, dname, sal from emp WHERE sal > 2000 and sal < 3000

op

fname	dname	sal
Smith	IT	2500

- ex : SELECT * FROM dept WHERE dept_no=20 AND dept_name = 'HR'

op

deptno	deptname	loc
20	HR	pune

Scanned with CamScanner

② OR LOGICAL OPERATOR

- OR operator is used to combine two or more conditions and
- it fetches the result which satisfy any one condition in OR statements.

ex SELECT fname, lname from emp
WHERE fname = 'miller' OR
lname = 'Ruth'

op

fname	lname
Smith	Ruth
miller	Ward

ex SELECT * FROM dept WHERE loc = 'Pune' OR deptname = 'IT';

deptid	dname	loc
31	IT	mumbai
42	Comp	Pune
63	HR	Pune

③ NOT Logical Operator

- Not operator is used to negate the condn & it fetches opposite of result which satisfy the condn
- It is used in combination with other keywords like NOT IN, NOT BETWEEN, etc

ex SELECT deptname, loc FROM dept
WHERE loc NOT IN ('chennai', 'Pune');

op

deptname	loc
IT	mumbai
HR	Binar

④ Comparison Operator

- Comparison operators (=, !=, <>, >=, <=, LIKE, BETWEEN, IN) are used in WHERE condn to fetch results from table.

⇒ Between Operator

- used to search for values that are within a set of values

ex SELECT fname, lname from emp
WHERE salary BETWEEN 2000 AND 3000;

op

fname	lname
Smith	Paul

Scanned with CamScanner

b) NOT IN operator

ex. SELECT fname from emp
WHERE ~~salary~~ loc NOT IN ('Pune', 'mumbai');

c) IN operator

ex fetches values from a set of literals.
- used to test whether or not a value is in the list of values provided after keyword IN.
- IN keyword can be used with any datatype in SQL

ex. SELECT fname, deptno from emp
WHERE job_id IN ('J1', 'J2');

d) >= , <= , > , < , =

ex SELECT fname from emp
WHERE salary >= 20000;

e) LIKE condition

- LIKE condition is used to perform wild card searches of valid search string values.
- search conditions can contain either characters or number

% ⇒ zero or many characters
_ ⇒ one character

ex 1) SELECT deptno, deptname
WHERE loc LIKE '%c'
-- location ends with c

c% ⇒ starts with c
h%i ⇒ starts with h, ends with c

ex 2) SELECT * from dept
WHERE loc LIKE 'mumb__';

ex 3) SELECT * from emp
WHERE fname NOT LIKE 'R%'
-- firstname should not start with R.

Scanned with CamScanner

Case Expression

- CASE is used to provide if-then-else type of logic to SQL.

Syntax :-

⇒
```
CASE col_name
WHEN condition1 THEN result1
WHEN condition2 THEN result2
ELSE result
END ;
```

ex :-

```
SELECT CASE (LOC)
WHEN 'Chennai' THEN 'TAMILNADU'
WHEN 'Thane' THEN 'MUMBAI'
ELSE 'NO IDEA'
END
FROM dept
```

- fetch data from 2 / more tables

SQL Joins

- A join clause is used to fetch data from 2 or more data tables based on join condition which have a common attribute.
- join clause (self join) is used to combine rows from one or more tables based on related column(s) between them.
- In SQL server, we have,

1) a) Self - Join

- A table is joined to itself in self join.

Syntax :-

```
SELECT <col_list>
FROM <table_name1> t1
JOIN <table_name1> t2
ON t1.column_name = t2.column_name
```

ex
```
SELECT t2.firstname as employee,
       t1.firstname as manager
FROM employee t1 JOIN employee t2
ON t1.emp_id = t2.mgr_id ;
```

O/p

employee	manager
Scott	Allen
Martin	~~Allen~~
Arthur	Jaki

Scanned with CamScanner

b) Inner Join

- Inner join fetches records that have matching values in both tables.

Syntax:-

```
SELECT <col_list>
FROM <table_name1> as t1
INNER JOIN <table_name2> as t2
ON t1.col_name = t2.col_name;
```

ex:-

emp

emp_id	fname	add
101	Rosh	Mumbai
102	Khush	Pune
103	Deep	Bihar
104	Geet	LA
105	Sant	NY
106	Priya	NULL

orders

orderid	empid	Order
1	104	P
2	102	Q
3	105	R
4	101	S
5	NULL	T
6		

- The 2 tables which we want to join must have a common column in them (here emp_id)

If we don't mention table name with emp_id, we'll get error bcoz system won't understand which emp_id is it (emp table or orders table)

```
SELECT emp.emp_id, orders.order_id, orders.Order
FROM emp
INNER JOIN orders
ON emp.emp_id = orders.emp_id
ORDER BY emp.emp_id;
```

op

emp_id	order_id	order
101	4	S
102	2	Q
104	1	P
105	3	R

⇒ Inner join query using alias. (table name alias)

```
SELECT e.emp_id, o.order_id, o.order
FROM emp as e
INNER JOIN orders o
ON e.emp_id = o.order_id
ORDER BY e.emp_id;
```

⇒ self join

Its mandatory to use alias while using self join

ex.

```
SELECT e1.emp_id, e2.l_name, e2.fname
FROM emp e1
INNER JOIN emp e2
ON e1.emp_id = e2.emp_id
ORDER BY e1.emp_id
```

↑ used for sorting.

⇒ Inner Join (column name alias)

```
SELECT d.Fname as FirstName,
       e.dept_no as Department
FROM dept d
INNER JOIN emp e
ON d.dept_id = e.dept_id
```

op ⇒

FirstName	Department
A	Acc
B	HR
C	IT
D	Comp

Scanned with CamScanner

② left Outer join

- The left outerjoin returns rows of left left (t1) even if there are no rows on right (t2) of the join clause.
- That ie, it Returns all those rows/records from left table and only mathing rows from right table.
- The result is NULL for rows on right table when there is no match
- Syntax:

⇒ SELECT < col_list >
FROM < table_name1 > as t1
LEFT OUTER JOIN < table_name2 > as t2
ON t1.colname = t2 col_name;

- ex:

emp

emp_id	Fname	add
101	A	P
102	B	Q
103	C	R
104	D	S
105	E	T
106	F	NULL

dept orders

orderid	empid	order
1	104	L
2	102	m
3	105	N
4	101	O
5	NULL	P

= SELECT e.emp_id, o.orderid, o.order
FROM emp e
LEFT OUTER JOIN orders o
ON e.emp_id = o.emp_id
ORDER BY e.emp_id.

o/p

emp_id	orderid	order
101	4	O
102	2	m
103	NULL	NULL
104	1	L
105	3	N
106	NULL	NULL

Thus, all records / rows is fetched from left table (emp) and only matching records from right table, when we dont have a matching record ex for emp_id 103, no data was there in order table, Thus it is set to NULL.

Scanned with CamScanner

d) Right Outer join

- The right outer join returns the rows of right table (t2) even if there are no matching rows on left table (t1)
- that is, it fetches all records of right table, and only matching records from left table. when there's no match, value is replaced with NULL
- The result is NULL for rows on left table when there is no match
- Syntax :

⇒
```
SELECT <col_list>
FROM <table_name1> as t1
RIGHT OUTER JOIN <table_name2> as t2
on t1.colname = t2.colname
```

- ex:

```
SELECT e.emp_id, o.order_id, o.order
FROM emp e
RIGHT OUTER JOIN order o
ON e.emp_id = o.emp_id
ORDER BY o.order_id desc;
```

o/p

emp_id	order_id	order
NULL	5	P
101	4	O
105	3	N
102	2	M
104	1	L

- Thus all records from right table (order) is fetched, & only matching records from left table (emp) when no match is found, val is set to NULL.

Scanned with CamScanner

e) FULL OUTER JOIN

- full outer join keyword return all records when there is a match in either left (t1) or right (t2) table records
- If there are rows in t1 that do not have matches in t2 or vice versa, those rows be listed as well.
- Syntax :

```
SELECT < col_list >
FROM < table-name1 > as t1
FULL OUTER JOIN < tablename2 > as t2
ON t1.coname = t2. Colname
```

- That is all records from both tables are fetched, & when there's no match of one table record into another, that record is listed as well with NULL value

ex

```
SELECT e.empid, o.orderid, o.order
FROM emp e
FULL OUTER JOIN Orders o
ON e.empid = o.order_id
ORDER BY e.emp_id;
```

o/p

emp_id	order_id	order
NULL	5	P
101	4	o
102	2	m
103	NULL	NULL
104	1	L
105	3	N
106	NULL	NULL

Scanned with CamScanner

5) cross join

- It displays all rows & all columns of both tables
- This is also called cartesian product
- Syntax:-

```
SELECT <col_list>
FROM <table_name> as t1
CROSS JOIN <table_name2> as t2;
```

- ex:-

```
SELECT e.emp_id, o.order_id, o.order
From emp e
CROSS JOIN orders O
ORDER BY e.emp_id
```

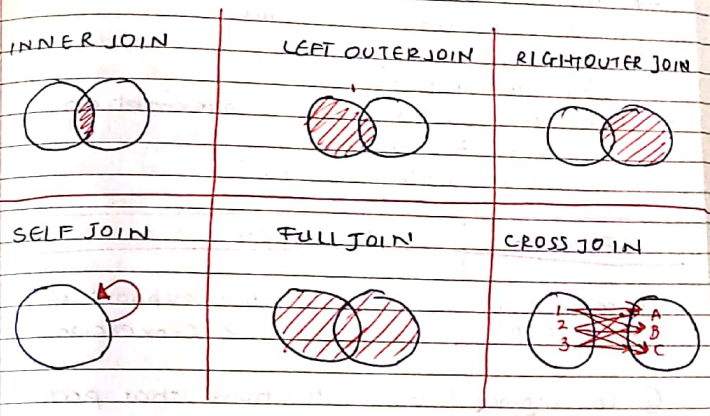
O/p

emp_id	order_id	order
101	1	L
101	2	M
101	3	N
101	4	O
101	5	P
102	1	L
102	2	M
102	3	N
102	4	O
102	5	P
⋮	⋮	⋮
106	1	L
106	2	M
106	3	N
106	4	O
106	5	P

6 rows in emp × 5 rows in orders

$6 \times 5 = 30$ ROWS in result

SQL JOINS



Scanned with CamScanner

SQL Built In functions

→ MS SQL built-in functions take 0 or more inputs & returns a value.

→ Built-in functions :
- Gets system Related information
- Used for calculations
- Manipulate input data

Function	Description
① Conversion function	functions that support data type casting & Conversion
② Logical functions	Scalar functions that perform logical operations
③ Math function	Scalar functions perform calculation usually based on i/p values provided as arguments & returns numeric value
④ Aggregate function	Aggregate fn perform calc on set of values & return a single value except for COUNT, aggregate fns ignore null values. They are used with GROUP BY clause of SELECT statement
⑤ String functions	Scalar fn performs operation on string i/p & returns string / numeric values
⑥ Date fn	fn that manipulate Dates

TRIGGERS

- A trigger is a special kind of stored procedure that automatically executes when an event occurs in database server
- DML trigger execute on a user tries to modify data through DML event
- DML events are INSERT, DELETE, UPDATE statements on a table / view

- Syntax :

```
CREATE [OR ALTER] TRIGGER schemaname.triggername
ON {Table | view}
FOR | ALTER | INSTEAD OF
[INSERT] [,] [UPDATE] [,] [DELETE]
AS
SQL Statements;
```

ex :-

```
CREATE TRIGGER Safety
ON DATABASE
FOR
create_table, alter_table, drop_table
as
print "you can not create, drop, alter
       table in this database"
roll back
```

Scanned with CamScanner

GROUP BY Clause

- group by clause is often used with aggregate fns (MAX, SUM, AVG) to group results from one or more columns
- used with SELECT statement to arrange required data into groups
- It groups rows that have same values
- It is used after WHERE clause
- ex

```
SELECT COUNT (salaries) AS count_sal, emp
FROM emp
GROUP BY salaries
```

HAVING clause

- while group by clause groups rows that have same values into summary rows.
- The having clause is used with WHERE clause to find rows with certain condn.
- ex.

```
SELECT COUNT (salaries) AS COUNT_SAL,
emp_id
FROM emp
GROUP BY salaries
HAVING COUNT(salaries) > 1;
```

Scanned with CamScanner

DCL Commands

- used to grant & take back authority from any database user.

a) GRANT : used to give user access privileges to database

ex:
GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

b) Revoke : used to take back permissions from user

ex:
REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2

TCL Commands → used to manage transactions in database
→ used to manage changes made by DML commands

- used only with DML Commands like INSERT, UPDATE, DELETE
- These operations are automatically committed in database that's why they cannot be used while creating or dropping tables

a) commit

- used to save any transactions to a database permanently
- Syntax : commit ;
- ex : DELETE FROM emp;
WHERE age < 18 ;
commit ;

b) Rollback

- restores database to last committed state
- used to undo transactions that have not already been saved to database
- Syntax : ROLLBACK;
- ex : DELETE FROM emp
WHERE age < 18;
ROLLBACK;

Scanned with CamScanner

SDS Page No
Date

c) **Savepoint** → used to temporarily save a transaction so that you can rollback to that point whenever necessary

- used to roll the transaction back to a certain point without rolling back the entire transaction
- Syntax: SAVEPOINT savepoint_name;
- ex: