SQL Complete Notes

INDEX

1) SELECT

- used to select data from a db

- select c1, c2
  from table;

2) SELECT DISTINCT

- used to return only distinct values

- select distinct c1
  from table;

3) WHERE

- used to filter records
- extracts only those records that fulfill a specified cond^{n}

- select c1, c2
  from table
  where condition;

4) ~~Conditional~~ Logical operators

- AND
- OR
- NOT

- where clause can be combined with AND, OR, NOT

- select * from customers
  where country = "Germany" AND city = "Berlin";

- select * from customers
where city = "Berlin" or city = "Munchen";

- select *
from table
where NOT condition;

eg. select * from customers
where NOT country = "Germany";

Combining AND, OR and NOT

select * from customers
where country = "Germany
AND (city = "Berlin" or city = "Munchen");

5) ORDER BY
- used to sort result-set in ascending or descending order using keywords asc or desc
- by default, asc

select * from table
order by asc | desc;
↓
one of these

ORDER BY several columns:
- select * from customers
order by country, customerName;
↓
order by country
if some rows have same country, then order by name

Page No.
Date

```
select * from customers
order by country asc, customerName desc;
```

country	customerName
A	C
A	A
B	Z
B	Y

6) INSERT INTO

- specify both column names & values to be inserted

```
- insert into student (ID, Name, Address)
  values (26, "Nikita", "ABC");
```

- if you are adding values for all the columns, no need to specify column names
- make sure to enter values in same order as columns

```
- insert into student
    values (26, "Nikita", 999999999, "ABC");
```

7) NULL values

- field with a null value is a field with no value
  eg. a new joiner is not assigned any project so his project ID = NULL

- can't test for NULL values with comparison operators
  = < <= >= > <> ~~not equal to~~

- we use IS NULL and IS NOT NULL operators instead

```
select * from customers
where address IS NULL;

select * from customers
where address IS NOT NULL;
```

8) UPDATE

- used to modify existing records in table

```
- update table
  set c1 = value1, c2 = value2
  where condition;
```

eg- update customers
set Name = "John", city = "London"
where ~~customer~~ customer ID = 1;

9) DELETE

- used to delete existing records in table

```
- delete from table
  where condition;
```

To delete all records:

- delete from customers;

↳ all rows will be deleted
table structure, columns will be intact

10) SELECT ~~TOP~~ LIMIT

- used to specify no of records to return

```
select * from table
where condition
limit number;
```

11) MIN() & MAX() functions

MIN() → returns smallest value of selected column

MAX() → returns largest value of selected column

```
select MIN(Price) AS SmallestPrice
from customers;
```

```
select MAX(Price) AS LargestPrice
from customers;
```

12) COUNT(), AVG(), SUM() fn=

```
select COUNT(ProductID)        → NULL values not counted
from Products;
```

```
select AVG(Price)              → NULL values are ignored
from Products;
```

```
select SUM(quantity)           → NULL values are ignored
from OrderDetails;
```

13) LIKE operator (NOT LIKE)

→ used in a where clause to search for a specified pattern in a column

```
select * from Customers
where name LIKE "a%";
```
↳ starts with 'a' followed by any no of char.

14) Wildcard characters

- it is used to substitute one or more characters in a string
- these are used with LIKE operator

→	%	zero or more char
→	_	single char
→	[]	any single char within brackets
→	^	any character not in brackets
→	-	any single char within range

bl% finds black, blue
h_t finds hit, hot, hat
h[oa] finds hot, hat
h[^oa] finds hit
c[a-b] finds cat, cbt

15) IN operator (NOT IN)

- allows you to specify multiple values in a where clause
- it is a shorthand for multiple OR conditions

- select c1, c2
from table
where c1 in (value1, value2, value3);

OR

- select c1, c2
from table
where c1 in (SELECT statement)
↓
this is sub-query

eg-
select * from customers
where country in (select country from Suppliers);

select * from customers
where customers in ('Germany', 'France', 'UK');

16) BETWEEN operator (NOT BETWEEN)
- selects values within a given range
↓
can be no, text, date

- begin and end values in the range are included

- select * from table
where c1 between v1 and v2;

Page No.
Date

17) Aliases
- used to give table or column a temporary name
- used to make column names more readable
- an alias only exists for duration of that query
- created with 'AS' keyword

- select c1 AS aliasName
from table;

- select c1, c2
from table AS aliasName;

eg- select Name, Address + ',' + PostalCode + ' ' +
City + ',' + Country AS address
from customers;

o/p

Name	address
Ajay	C-115, 110042, Delhi, India
Ali	B-15, 110085, Patna, India

eg- select p.orderID, c.customerName
from customers AS c, products AS p
where p.customerID = c.customerID;

- aliases can be useful when:
- more than one table involved in a query
- functions are used in the query
- column names are big or not readable
- two or more columns are combined together

- if alias is one word, not compulsory to write "AS"
- if more than one word, compulsory to write "AS"

18) UNION operator

- used to combine result set of two or more SELECT statements

- every SELECT statement within UNION must have same no of columns

- columns must have similar data type

- columns in every SELECT statement must be in same order

- select c1, c2 from table1
  UNION
  select c3, c4 from table2;

- displays only distinct values

eg. select city from customers
    UNION
    select city from Suppliers
    order by city;

o/p → above query displays unique cities from both tables combined

City
Chandigarh
Jaipur
Patna
Raipur
Trivendram

Page No.
Date

UNION ALL operator

- this returns duplicates values as well from the union of two tables
- this way, there is no loss of data

T1 city	country
A	G
Z	B
Y	G
P	G

T2 city	country
Q	B
A	B
X	G

```
select city from T1
UNION ALL
select city from T2
order by city;
```

o/p city

A
A
P
Q
X
Y
Z

} duplicate entries are also printed

UNION with WHERE

```
eg- select city, country from T1
    where country = 'G'
    UNION
    select city, country from T2
    where country = 'G';
```

city	country
A	G
Y	G
P	G
X	G

- column names of output will be same as first select statement

eg-

```
select 'customer' as Type, city, country
from customers
UNION
select 'supplier', city, country
from suppliers ;
```

Customers

city	country
A	X
B	Y

Suppliers

city	country
A	Z

o/p

Type	City	country
customer	A	X
customer	B	Y
supplier	A	Z

→ if we put order by city asc in above query, output:

Type	City	country
customer	A	X
supplier	A	Z
customer	B	Y

Page No.
Date

19) GROUP BY statement

- it groups rows that same values in one attribute value

→ eg- find no of customers in each country

this means, we should group data using country column

ans- select count(customerID), country
from Customers
GROUP BY country;

Table →

customerID	country
1	A
2	B
3	A
4	C
5	C
6	D

o/p →

count(customerID)	country
2	A
1	B
2	C
1	D

→ eg- select count(customerID), Country
from customers
GROUP BY country
order by count(customerID) desc;

- group by statement is often used with aggregate fn^s i.e COUNT(), MAX(), MIN(), SUM(), AVG() to group the result set by one or more columns.

Page No.
Date

o/p →

count(customerID)	country
2	C
2	A
1	D
1	B

Note: we have put order by on first column
If two rows have same value in first column, we order by in descending order, based on second column (country)

GROUP BY with JOIN

eg.
```
select s.name, count(o.orderID)
from orders o
left join shippers s
on o.shipperID = s.shipperID
group by name;
```

shippers

shipperID	name
1	A
2	B
3	C

orders

order ID	shipper ID
1	2
2	3
3	3
4	2
5	2
6	3

o/p

name	count(o.orderID)
A	0
B	3
C	3

Page No.
Date

20) HAVING clause

- it was added to SQL bcoz WHERE keyword can't be used with aggregate functions

- select *
  from table
  where condition
  GROUP BY C1
  HAVING condition
  ORDER BY C2
  LIMIT N
  OFFSET M;

eg- list no of customers in each country in desc order
Only include countries with more than 5 customers

ans - select count(custID), country
from customers
group by country
having count(custID) > 5;
order by count(custID) desc;

21) EXISTS operator

- used to test existence of any record in a subquery
- returns TRUE if subquery returns one or more records

- select * from table
WHERE EXISTS
(select C1 from table where condition);

eg- ~~select name from suppliers s~~
~~where~~

eg- suppliers

supplierID	sname
1	A
2	B
3	C

products

pname	price	supplierID
X	18	1
Y	20	2
Z	22	3
M	15	1
N	21	2
P	19	3

Print list of suppliers with a product price < 20

ans → select sname
from suppliers
where EXISTS (select pname from products
where products.supplierID =
~~product~~ suppliers.supplierID and price < 20);

o/p →

sname
A
C

2) ANY operator and ALL operator

- these allow you to perform a comparison b/w a single column value and a range of other values

Page No.
Date

ANY :

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the cond^n

ANY means that the cond^n will be true if the operation is true for any of the values in the range.

- select c1, c2
  from table
  where
  c3 operator ANY (select c
  from table
  where condition);

a std comparison operator
= <> > >= <= <

eg- ~~select pname~~
~~from products~~

eg- list the product names if it finds any records in the order table that has quantity equal to 10.

Product

PID	Name
1	A
2	B
3	C
4	D
5	E

Order

PID	qty
1	8
2	10
3	7
4	9
5	10

ans → select Name
from Product
where PID = ANY (select PID from Order
where qty = 10);

o/p

name
B
E

ALL

→ returns a boolean value as a result
→ returns TRUE if ALL of the subquery values meet the condition
→ is used with select, where, having statements

ALL means that condition will be true only if operation is true for all values in range

syntax with SELECT:

```
select ALL column
from table
where condition;
```

syntax with WHERE or HAVING:

```
select column
from table
where column operator ALL (select column
                           from table
                           where condition);
```

eg- list all product names.

ans → select ALL name
from products;

eg- list the product name if ALL the records in ~~Reco~~ Order table has qty equal to 10.

ans → select pname
from product
where PID = ALL
(select PID from order where qty = 10);

o/p → | pname |

∵ qty column has many different values, not just 10

23) SELECT INTO statement

- copies data from one table into a new table

- SELECT *
INTO newtable [in externaldb]
FROM oldtable
WHERE condition;

copy all columns into new table

optional

- SELECT c1, c2
INTO newtable [in externaldb]
FROM oldtable
WHERE condition;

copy only some columns into new table

(1) eg- to create backup copy of customers table :
- select * into CustomersBackup2017
from customers;
~~where TRUE~~;

(2) eg- to copy customers table into a new table in another database:

- select * into CustomersBackup2017 in 'Backup.mdb'
  from customers;

(3) eg - to copy data from more than one table into a new table:

- select customers.cname, orders.OID
  INTO customerOrderBackup2017
  from customers
  LEFT JOIN orders
  ON customers.CID = order.CID;

* SELECT INTO can be used to create a new & empty table using schema of an existing table. Just add a WHERE clause that causes the query to return no data (i.e entries of existing table won't be copied in new one)

⇒ select *
  INTO newtable
  from oldtable
  where 1 = 0;

24) INSERT INTO SELECT statement

→ copies data from one table and inserts it into another table
→ data types in source & target tables should match
→ existing records in target table are unaffected

→ insert into table2
select * from table1
where condition;

→ to copy only some columns from one table into another table:

→ insert into table2 (c1, c2, c3)
select c4, c5, c6
from table1
where condition;

eg- Copy suppliers into customers
→ insert into customers
select * from suppliers

↳ columns that are not filled with data will contain NULL

25) CASE statement

→ it goes through conditions and returns a value when the first condition is met
→ it is like an if-else statement
→ once a condition is true, it will stop reading and return the result.
→ if no conditions are true, it returns value in ELSE clause (optional)
→ if there is no ELSE part and no conditions are TRUE, it returns NULL

→ CASE
WHEN condition1 THEN result1
WHEN condition2 THEN result2
ELSE result
END;

eg- Select orderID, qty
CASE
WHEN qty > 30 THEN "qty is greater than 30"
WHEN qty = 30 THEN "qty is 30"
ELSE "qty is under 30"
END AS QuantityText
from order;

o/p → orderID qty QuantityText

eg- order the customers by city
if city is NULL, then order by country

ans- select * from customers
order by
(CASE
WHEN city IS NULL THEN country
ELSE city
END);

2) IFNULL() function

- it lets you return an alternative value if an expression is NULL

eg- select name, UnitPrice * (qty + IFNULL(unitsonorder, 0))
from products

↓
used bcoz we had to make a calculation, which can't take place with NULL values

Page No.
Date

27) Stored Procedure

→ it is a prepared SQL code that you can save, so that code can be reused over and over again

→ So if you have an SQL query that you write over and over again, save it as a stored procedure and then just call it to execute it

→ You can also pass parameters to a stored procedure, so that stored procedure can act based on the parameter value that is passed.

→ syntax:

```
CREATE PROCEDURE pname
AS
sql_statement
GO;
```

→ to execute a stored procedure:

```
EXEC pname;
```

eg.

```
create procedure selectAllCust
AS
select * from customers
GO;

EXEC selectAllCust;
```

Page No.
Date

eg- create procedure selectAllCust @city varchar(30)
AS
select * from customers where city = @city
GO;

EXEC selectAllCust @city = "London";

* Illy, we can declare multiple parameters & pass them using EXEC statement separated by commas

28) Comments
- single line --
- multiple lines /* */

29) Arithmetic operators

+	add
-	subtract
*	multiply
/	divide
%	remainder

30) Bitwise operators

&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR

31) Comparison operators

=	
>	
<	
<=	
>=	
<>	

32) Compound operators

+=	/=
-=	%=
*=	&=

^-= bitwise exclusive equals
&= bitwise AND equals
|*= bitwise OR equals

33) Logical operators

ALL

AND

ANY

BETWEEN

EXISTS

IN

LIKE

NOT

OR

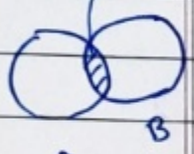
Join = cross product + cond^n

Page No.
Date

Types of join
- Natural
- Inner
- Self
- Equi
- Outer → left, right, full

Natural join

when we have to make value of common attribute "equal"



Emp

ENo	EName	Address
1	Ram	Delhi
2	Varun	Chd
3	Ravi	Chd
4	Amrit	Delhi

Dept

DNo	Name	ENo
D1	HR	1
D2	IT	2
D3	MKT	4

→ Display employees working in a dept

Select EName from (Emp, Dept) → cross product
where Emp.ENo = Dept.ENo;

~~ENo EName D~~

selecting these rows from Emp × Dept

desired ans

1	Ram	D1	1
2	Varun	D2	2
4	Amrit	D3	4

=> select EName from Emp Natural Join Dept;

Page No.
Date

Emp x Dept rows = 4X3

ENO	EName	DNo	ENo
1	Ram	D1	1
		D2	2
		D3	4
2	Varun	D1	1
		D2	2
		D3	4
3	Ravi	D1	1
		D2	2
		D3	4
4	Amit	D1	1
		D2	2
		D3	4

Self Join

Student (Sid) Course (Cid)

Sid	Cid	Since
S1	C1	2016
S2	C2	2017
S1	C2	2017

Study

Sid, Cid → P.K of study

We want to take cross product of study with itself. That's why called self join

Display Student ID of students who are enrolled in atleast two courses

→ select T1.Sid from Study as T1, Study as T2 where T1.Sid = T2.Sid and T1.Cid <> T2.Cid;

~~S1 X S2~~

T1 X T2

Sid	Cid	Sid	Cid	
S1	C1	S1	C1	
S1	C1	S2	C2	
S1	C1	S1	C2	→ o/p
S2	C2	S1	C1	
S2	C2	S2	C2	
S2	C2	S1	C2	
S1	C2	S1	C1	
S1	C2	S2	C2	
S1	C2	S1	C2	

Equi Join

join with equal cond^2

Emp

ENo	EName	Address
1	Ram	Delhi
2	Varun	Chd
3	Ravi	Chd
4	Amit	Delhi

Dept

DNO	Location	ENO
D1	Delhi	1
D2	Pune	2
D3	Patna	4

Display name of employee who worked in a dept having location same as their address

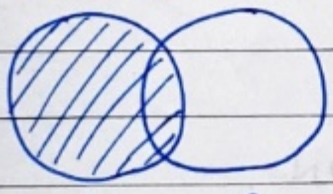
→ select EName from Emp, Dept
where Emp. ENo = Dept. ENo and
Emp. Address = Dept. location

o/p →

ENo	EName	Address	Location	ENo
1	Ram	Delhi	Delhi	1
1	Ram	Delhi	Pune	2
1	Ram	Delhi	Patna	4
2	Varun	Chd	Delhi	1
2	Varun	Chd	Pune	2
2	Varun	Chd	Patna	4
3	Ravi	Chd	Delhi	1
3	Ravi	Chd	Pune	2
3	Ravi	Chd	Patna	4
4	Amrit	Delhi	Delhi	1
4	Amrit	Delhi	Pune	2
4	Amrit	Delhi	Patna	4

left outer join

= Natural join + rows which are in left table but not right table



Emp

ENO	EName	DNO
E1	Varun	D1
E2	Amrit	D2
E3	Ravi	D1
E4	Nitin	-

Dept

DNO	DName	Location
D1	IT	Delhi
D2	HR	Hyd
D3	Fin	Pune

Criw employee & their respective dept details for all employee

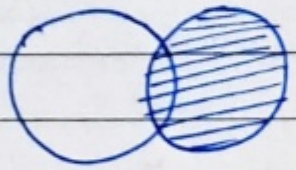
→ select ENo, EName, DName, Location from Emp left outer join Dept on Emp.DNo = Dept.DNo;

o/p

ENo	EName	DName	Location
E1	Varun	IT	Delhi
E2	Amit	HR	Hyd
E3	Rani	IT	Delhi
E4	Nitin	-	-

3 Right outer join

= Natural join + rows which are in right table but not in left table



Display details of all dept & also the name of employee working in each dept.

→ select DNo, DName, Location, EName
from Emp
Right Join Dept
on Emp.ENo = Dept.DNo;

* natural join = inner join (syntax diff.)
* left outer join = left join (syntax diff)
* right " " right " (syntax diff.)

JOIN

Page No.
Date

Q2,3,4 after join notes

Manager

mgr_id	mgr_name	city
M101	Ajay	Pune
M102	Amit	Mumbai
M103	Sandeep	Banglore
M104	Prakhar	Pune
M105	Jaydeep	Indore
M106	Shubham	Banglore

Department

dept_id	dept_name
1	Admin
2	HR
3.	Developer
4	Tester
5	Accounts

Employee

Field	~~emp_id Type~~ ~~emp_name salary city~~
emp_id	
emp_name	
salary	
city	
dept_id	
mgr_id	

Page No. _____
Date

Employee

emp_id	emp_name	salary	city	dept_id	mgr_id
1	Karan	300000	Mumbai	2	M102
2	Rohit	75000	Banglore	1	M106
3	Ankush	35000	Ahmdbd	5	M105
4	Priyanshi	500000	Kolkata	3	M102
5	Sanket	100000	Pune	3	M101
6	Shruti	80000	Indore	4	M105
7	Jayraj	75000	Bhopal	2	M103
8	Dilip	66000	Mumbai	3	M102
9	Geet	17000	Mumbai	2	M101

1) WAQ to find name of dept where more than 2 employees are working.

→ select dept_name
from dept left outer join emp
on dept.dept_id = emp.dept_id
group by ~~count~~ {dept_id}
having count (dept_id) > 2;

using left outer join

OR

→ select dept.dept_name
from dept left join emp
on dept.dept_id = emp.dept_id
group by dept.dept_id
having count (dept.dept_id) > 2;

using left join

OR → select d.dept_name
from dept d left join emp e
on d.dept_id = e.dept_id
group by d.dept_id
having count (d.dept_id) > 2;

using alias

Page No.
Date

Practise question on JOIN

2) WAQ to calculate avg salary of each dept which is higher than 75000. Also display dept name in descending order.

→

```
select avg(e.salary), d.dept_name
from emp e left join dept d
on d.dept_id = e.dept_id
group by d.dept_id
having ~~count~~ avg(e.salary) > 75000
order by d.dept_name desc;
```

3) WAQ to find managers & employees who belong to same city.

→

```
select e.emp_name, m.mgr_name, e.city
from manager m right join emp e
on e.mgr_id = m.mgr_id
where e.city = m.city
```

4) WAQ to find those emp names whose salary ∈ (35000, 90000). Also print their dept & mgr name.

→

```
select d.dept_name, m.mgr_name, e.emp_name, e.salary
from emp e inner join dept d
on e.dept_id = d.dept_id inner join manager m
on m.mgr_id = e.mgr_id
where e.salary between 35000 and 90000;
```

Index

1) MySQL functions
   ↳ string functions
   numeric functions
2) Views
3) Transactions
4) Roles
5) Triggers
6) Constraints

MySQL functions

Page No. _____
Date | | | |

String functions

1) ASCII — returns ASCII value of specific char
2) CHAR_LENGTH
3) CHARACTER_LENGTH } returns length of string (in chars)
4) CONCAT — adds two or more expressions together
5) CONCAT_WS — adds two or more exp: together with a separator

eg- select CONCAT_WS ("-", "Nikita", "Deswal");
o/p → Nikita-Deswal

eg- select CONCAT ("Nikita", "Deswal");
o/p → Nikita Deswal

eg- select CHAR_LENGTH ("Nikita");
6

6) FIELD — returns index position of a value in a list of values

syntax → FIELD (value, V1, V2, V3 ---)
↓ value to be searched
V1, V2, V3 ---) → list to search value from

eg- select FIELD ("q", "s", "q");
o/p → 2

7) FIND_IN_SET — returns position of string in list of strings

syntax → FIND_IN_SET(string, string_list)

eg- select FIND_IN_SET ("a", "s,q,l");
o/p → 0

8) INSERT — inserts a string within a string, at specified position and for certain no of chars

INSERT(string, position, number, string2)

string → will be modified
position → position where to insert string2
number → no of chars to replace
string2 → string to be inserted in 'string'

eg- INSERT("W3Schools.com", 1, 9, "Example")
→ Example.com

9) INSTR returns position of first occurrence of a string in another string
INSERT (string1, string2)
↙ string that will be searched in
↘ string to search for in string1

eg- SELECT INSTR("W3Schools.com", "COM");
o/p → 11

10) LCASE converts a string to lower case
LCASE(SQL) => sql

11) UCASE converts a string to upper case
UCASE(sql) => SQL

12) LEFT extracts a no of chars from a string, starting from left
LEFT ("SQL Tutorial", 3) => SQL

13) RIGHT extracts a no of chars from a string, starting from right
RIGHT ("SQL is cool", 4) => cool

14) LENGTH returns length of string in bytes
LENGTH("Nikku") => 5

15) LOWER — same as LCASE()
LOWER ("SQL") => sql

16) UPPER — same as UCASE()
UPPER ("sql") => SQL

17) MID — extracts a substring from a string
same as SUBSTR() and SUBSTRING()
MID (string, start, length)
MID ("Nikita", 1, 4) => Niki

18) POSITION — returns position of first occurrence of substring in a string
case-insensitive i.e s ≡ S
POSITION (substr IN str)
POSITION ("i" IN "NIKITA") => 2

19) REPEAT — repeats a string as many times as specified
REPEAT (str, number)
REPEAT ("x", 3) => xxx

20) REPLACE — replaces all occurrences of a substr within a str, with a new substr
REPLACE (string, from_str, new_str)
↓ original str ↓ substr to be replaced ↘ new replacement substr
REPLACE ("SQL tutorial", "SQL", "HTML")
=> HTML tutorial

21) REVERSE — reverses the string & returns the result
REVERSE ("MAM") => MAM

Page No.
Date

22) STRCMP — compares two strings

STRCMP (S1, S2)

S1 < S2 fn returns -1

S1 = S2 fn returns 0

S1 > S2 fn returns 1

STRCMP ("SQL", "HTML") => 1

23) SUBSTR() SUBSTRING() → same as MID()

24) TRIM() ← not so necessary fn, google if interested

SPACE()

LTRIM()

RTRIM()

LPAD()

RPAD()

FORMAT()

Numeric Functions

Page No.
Date

1) ABS · returns absolute value of a no
ABS(-5) => 5

2) AVG returns avg of a column values
SELECT AVG(Price) from customers;

3) CEIL or CEILING ≥ no
CEIL(23.4) => 24

4) FLOOR ≤ no
CEIL(23.4) => 23

5) COUNT select count(SID) from customers;
↓
counts SID cdm values

6) DEGREES convert radian value into degrees
DEGREES(1.5) => 85.94...

7) DIV used for division
SELECT 10 DIV 5; -) 2

8) EXP EXP(2) => e^2
EXP(1) => $e^1 = e = 2.718...$

9) GREATEST GREATEST(3, 12, 14, 8) => 14

10) LEAST LEAST(3, 12, 14, 8) => 3

11) MAX

12) MIN

13) MOD modulus operator 10%2 = 0

Page No.
Date

14) POW
or
POWER
POW (2,4) = 16

15) ROUND
round off
ROUND (15.6) => 16
ROUND (15.6, 2) => 15.60
ROUND (15.6, 0) => 16.0

16) SIGN
SIGN (15) => 1
SIGN (-15) -> -1
SIGN (0) -> 0

17) SQRT

18) SUM

19) TRUNCATE
return a no truncated to n decimals
TRUNCATE (12.752, 2) => 12.75

many more fn's
can read from W3Schools

VIEWS

- These are kind of virtual tables
- Vreiw also has rows & columns, as in real table in the db
- We can create a view by selecting fields from one or more tables present in the db
- Vreiw can either have all rows of a table or specific rows based on certain condition

Creating views

- from single table
- from multiple tables

syntax:

```
CREATE VIEW viewName AS
select * from table where condition;
```

```
CREATE VIEW viewName AS
select table1.column1 , table2.column2
from table1, table2
where table1.column1 = table2.column1 ;
```

to display data of a view:

```
select * from viewName;
```

deleting a view

```
DROP VIEW viewName;
```

Page No.
Date

updating views

following cond^{n} should be satisfied to be able to update a view:

1) SELECT statement used to create view should not include GROUP BY or ORDER BY clause.
2) SELECT statement should not have distinct keyword
3) view should have all NOT NULL values
4) view should not be created using nested or complex query
5) created from single table

```
CREATE OR REPLACE VIEW viewName AS
select * from table where condition;

INSERT INTO viewName (c1, c2, c3)
values (v1, v2, v3);
```

Deleting row from view

```
DELETE from viewName
where condition;
```

WITH CHECK OPTION clause

- very useful clause for views
- applicable to an updated view
- used to prevent insertion of rows in view where condition in WHERE clause in CREATE VIEW is not satisfied
- If we have used WITH CHECK OPTION in CREATE VIEW statement, and if UPDATE or INSERT clause does not satisfy the conditions, then they will return an error.

eg- CREATE VIEW sample AS
select SID, Name
from student
where Name is not null
WITH CHECK OPTION;

Uses of a view

A good db should contain views bcoz:

1) Restricting data access:
Views provide an additional level of table security by restricting access to a predetermined set of rows & columns

2) Hiding data complexity:
A view can hide the complexity that exists in a multiple table join.

3) Simply commands for user:

4) Store complex queries

5) Rename columns

6) Multiple view facility:
Different views can be created on one table for multiple users.

TRANSACTIONS

- Transactions group a set of tasks into a single execution unit
- Each transaction begins with a specific task and ends when all the tasks in the group successfully complete
- If any of the tasks fail, the transaction ends.
- ~~Trans~~ It has only two results
  - success
  - failure
- Incomplete steps result in failure of transaction
- A db transaction must has ACID properties i.e should be atomic, consistent, isolated, durable

Implementing transactions

Following cmds can't be used while creating tables and are used only with DML commands →
INSERT, UPDATE, DELETE.

1) BEGIN TRANSACTION
- it indicates start point of transaction
- BEGIN TRANSACTION trans;

2) SET TRANSACTION
- place a name on transaction
- SET TRANSACTION READ WRITE;
  OR, READ ONLY;

3) COMMIT
- If everything is in order with all statements within a single transaction, all changes are recorded together in db, called committed

- COMMIT command saves all transactions to the db since the last COMMIT or ROLLBACK command.
- COMMIT;

eg. to show to delete those records from table which have age = 20 and then COMMIT the changes in db

Student

RollNo	Age
1	20
2	21
3	18
4	20

```
delete from student where age = 20;
COMMIT;

select * from student;
```

RollNo	Age
2	21
3	18

4) ROLLBACK
- If any error occurs with any of SQL grouped statements, all changes need to be aborted. This process of reversing changes is called rollback.
- This cmd can only be used to undo transactions since last COMMIT or ROLLBACK and was issued.
- syntax: ROLLBACK;

eg. delete those records whose age = 20 and then ROLLBACK changes in db

```
- delete from student where age = 20;
  ROLLBACK;
  select * from student;
```

ROLLNO	Age
1	20
2	21
3	18
4	20

5) SAVEPOINT

- creates points within groups of transactions in which to ROLLBACK
- It is a point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.

- syntax:
  SAVEPOINT sname;
  ↓
- This cmd is used only in creation of SAVEPOINT among all transactions
  In general, ROLLBACK is used to undo a group of transactions

- syntax to rollback to savepoint cmd:
  ROLLBACK to sname;

  eg- SAVEPOINT SP1;
  delete from student where age=20;
  SAVEPOINT SP2;
  ROLLBACK TO SP1;

  o/p will have all student records bcoz we rolled back to SP1

6) RELEASE SAVEPOINT

- used to remove a savepoint that you have created
- syntax: RELEASE SAVEPOINT sname;
- once a savepoint has been released, you can no longer use rollback cmd to undo transactions performed since last savepoint

Page No.
Date

ROLES

- created to ease setup & maintainence of security model
- it is a named group of related privileges that can be granted to user
- If you define roles, you can:
  - You can grant or revoke privileges to users
  - You can create roles or use system roles pre-defined

System roles	Privileges granted to the role
- connect	create table, create view, create session etc.
- resource	create procedure, sequence, table, trigger etc.
	used to restrict access to db objects
- DBA	all system privileges
↳ db admin	

Creating & assigning a role

- First, DBA must create the role
- Then DBA can assign privileges to role and users to role.

- syntax: CREATE ROLE manager;
  name of role

- Now, DBA can use grant statement to assign users to the role as well as assign privileges to role
- Its easier to GRANT or REVOKE privileges to user through a role rather than assigning a privilege directly to every user.
- If a role is identified by a password, GRANT or REVOKE privileges have to be identified by the password.

- syntax to grant privileges to a role:
  GRANT create table, create view TO manager;

- syntax to grant a role to users:
  GRANT manager to SAM, JOHN;

- syntax to revoke privilege from a role:
  REVOKE create table from manager;

- syntax to drop a role:
  DROP ROLE manager;

TRIGGERS

- it is a stored procedure in db which automatically invokes whenever a special event in db occurs
- Eg- a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

- syntax:

```
CREATE TRIGGER tname
BEFORE ------ or AFTER
INSERT -- or UPDATE -- or DELETE
ON table_name
FOR EACH ROW
[ body of trigger ]
```

BEFORE → run the triggering action before triggering statement is run

AFTER → ——— after ———

Eg- Given students marks db. Create a trigger so that total and avg of specified marks is automatically inserted whenever a record is inserted.

Here, trigger will invoke before record is inserted so, BEFORE tag can be used.

```
create trigger stud_marks
before INSERT
on
student
for each row
set student.total = student.subj1 + student.subj2,
```

Page No. _____
Date

Student.per = Student.total * 60/100;

=) whenever subject marks are entered, before inserting this data into the db, trigger will compute total and avg values and insert all the data together in Student table.

- insert into Student values (100, "AB", 20, 20, 0, 0);
select * from Student;

sid	name	subj1	subj2	total	per
100	AB	20	20	40	24

CONSTRAINTS

1) NOT NULL → we can't store null value in column in which it is applied

eg-

```
create table student
(
  ID int(6) NOT NULL,
  NAME varchar(10) NOT NULL,
  ADDRESS varchar(20)
);
```

2) UNIQUE →

```
create table student
(
  ID int(6) NOT NULL UNIQUE
  :
  :
);
```

3) PRIMARY KEY → field which uniquely ~~dbt~~ identifies each row in the table
can't have null value
should be unique, otherwise shows error

basically, primary key = not null + unique

eg-

```
create table student
(
  ID int(6) ——,
  NAME —— ——,
  PRIMARY KEY (ID)
);
```

4) FOREIGN KEY → it is a field in a table which uniquely identifies each rows of another table
i.e this field points to P.K. of another table
It usually creates a kind of link b/w tables.

eg-

```
create table orders (
    O_ID int NOT NULL,
    ORDER_NO int NOT NULL,
    C_ID int,
    PRIMARY KEY (O_ID) REFERENCES customers (C_ID));
```

5) CHECK → using this, we can specify a cond° for a field, which should be specified at time of entering values for this field

eg-

```
create table student (
    ID int (6) NOT NULL,
    NAME varchar (10) NOT NULL,
    AGE ~~INT~~ int NOT NULL CHECK (AGE >= 18));
```

6) DEFAULT → to provide default value for fields
=) if at the time of entering values in table if user doesn't specify any value for these fields,
then the default value will be assigned to them

```
create table student (
    ID int NOT NULL, NAME varchar NOT NULL,
    AGE int DEFAULT 18);
insert into student values (25, "ABC");
```

↳ age of ABC will be 18