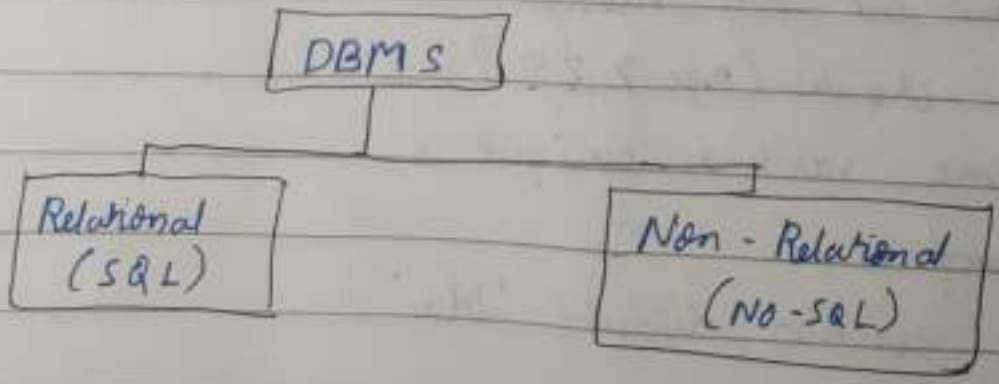
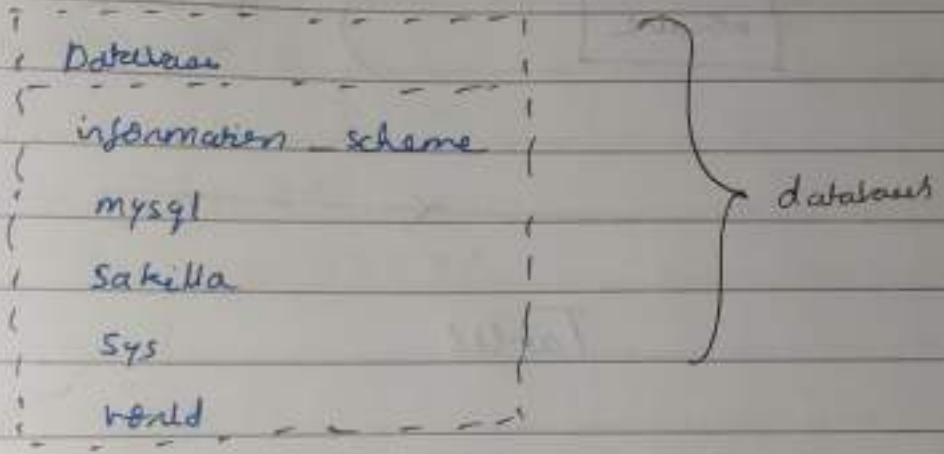
SQL

Def^{n} :- My sql is a relational database management system developed by oracle that is basic on structured query language (SQL).



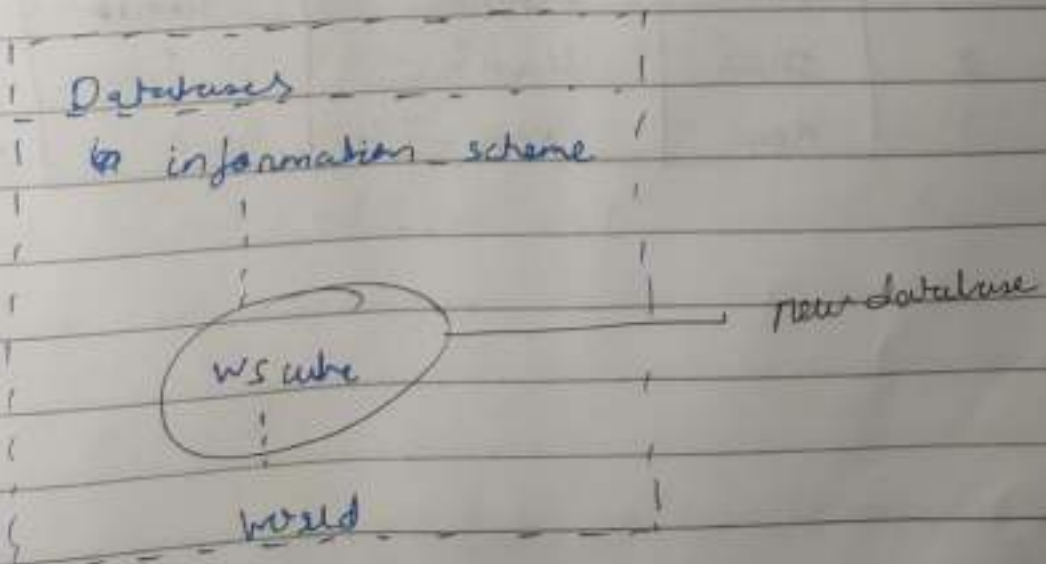
•> Show databases ——→ It shows all databases stored within a folder

Ex :- Show databases



•> Create database ——→ It creates a new database.

Ex Create database wscube



• use ⟶ it will enter into given database

Ex use wscube

Database changed

wscube

we will enter in wscube database

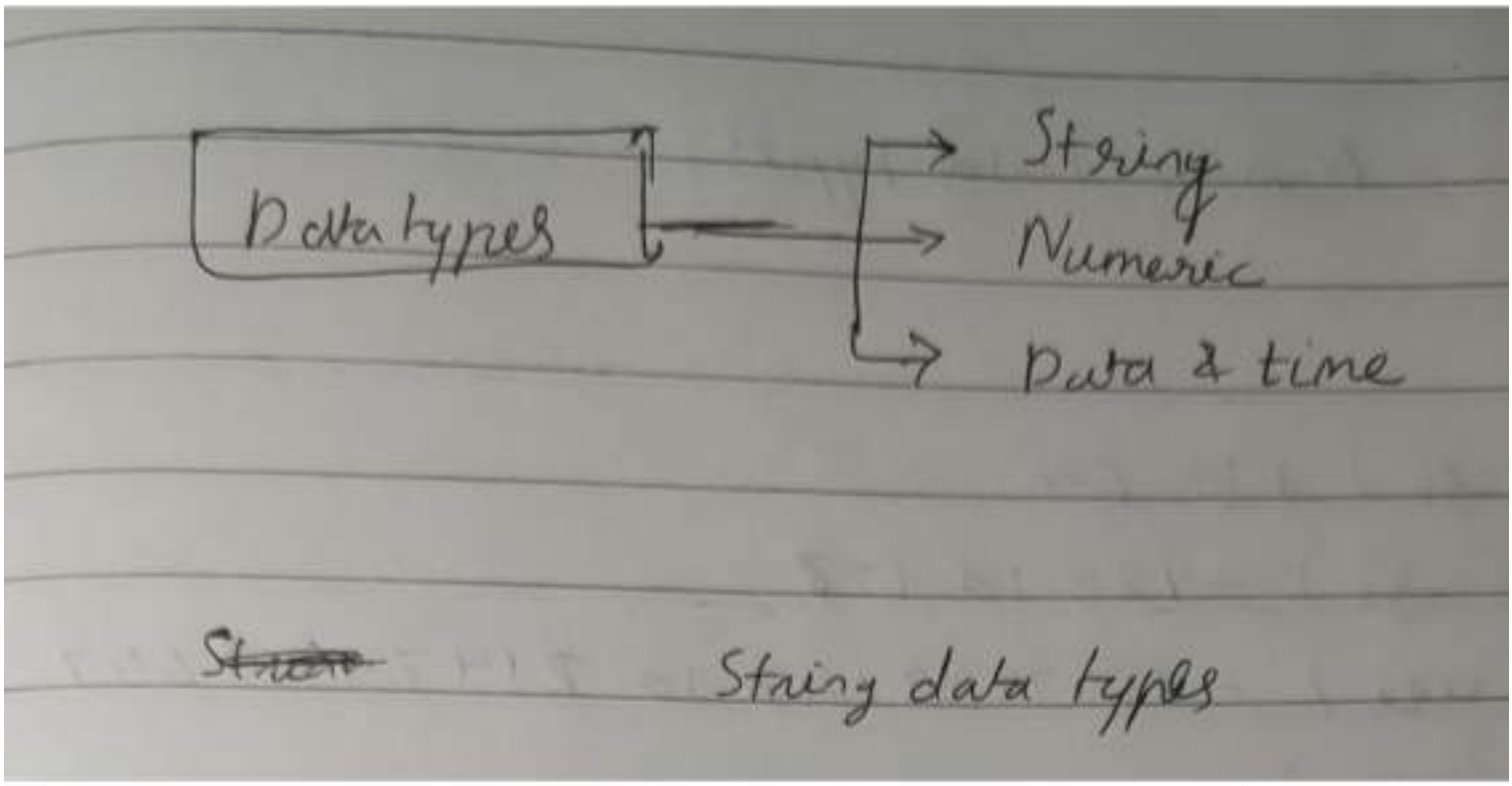
Tables

Def" :- A Database table is a collection of rows & columns that contains relational data.

columns ↓ ↓ ↓

Id	Name	Email	DOB	
1	Ram	ram - - -	2	←
2	Sham	Sham - - -	5	←
3	Rom	Rom - - -	6	←

Rows



~~String~~ String data types

~~Strin~~ String data types

~~char (size 2)~~

char (size) 0 to 255
varchar (size) 0 to 65535
Binary (size)
varbinary (size)
Tinytext 255 characters
Text (size) 65,535 bytes
Mediumtext 16,777,215 characters
longtext 4,294,967,295 characters
tinyblob 255 bytes
Blob (size) 65,535 bytes
mediumblob 16,777,215 bytes
longblob 4,294,967,295 bytes
Enum (val1, val2, val3, ...) upto 65535 values
Set (val1, val2, val3 ...) upto 64 values

gender can be put like "M" "F"

Date data types

Date '1000-01-01' to '9999-12-31'
Datetime (fsp) YYYY-MM-DD hh:mm:ss
Timestamp (fsp) hh:mm:ss
Time (fsp) hh:mm:ss
Year four-digit format: 1901

Tables

~~CREATE~~

```
CREATE TABLE Users ---- Create table Users
(
    id int unsigned, ---- "id"
    name varchar (100), ---- "name"
    email varchar (150), ---- "email"
    password varchar (100), ---- "password"
    contact varchar (15), ---- "contact"
    address text, ---- "address"
    dob date, ---- "dob"
    gender enum ("M", "F", "O"), ---- "gender"
    status boolean ---- "Status"
)
```

Input

id	name	email	Password	contact	address	dob	gender	status

Output

- Select * From ⟶ It show data stored in created ~~data~~ table.

Ex :-

```
SELECT * FROM wscube.users;
```

— It will show data stored in database "wscube" & table "users".

```
Empty set (0.00) sec
```

— It is empty as no data is inserted.

—— or ——

Insert Into ⟶ It inserts data into tables

Type -1

Ex :-

```
INSERT INTO Users
(id, name, email, password, contact, address, dob, gender,
 status)
VALUES
(1, "John", "John@gmail.com", "12345678",
 "98765432 1", "Jodhpur, Rajasthan", "1999-01-10",
 "M", 1)
;
```

Input

id	name	email	password	contact	address	dob	gender	status
1	John	John@gmail.com	12345678	9876543 21	Jodhpur Rajasthan	1999-01-10	M	1

output

~~Insert~~ Type 2 :-

Ex :-

```
INSERT INTO ~~USER~~ users
(id, name, email, password, contact, addres, dob,
 gender, status)
VALUES
```

Input

ID 1

```
(2, "Jenny", "Jenny@gmail.com", "123456789",
"9876543210", "Jodhpur, Rajasthan", "1999-01-10",
"M", 1),
```

ID 2

```
(3, "Bhagirath", "bhagirath@tech.com", "3452422343",
"3423123424", "Waratwada, Pune", "2002-11-08",
"F", 1)
;
```

id	name	email	password	contact	address	dob	gender	status
1	John	John@gmail.com	123456789	987654321	Jodhpur Rajasthan	1999-01-10	M	1
2	Jenny	Jenny@gmail.com	123456789	9876543221	Jodhpur Rajasthan	1999-01-10	M	1
3	Bhagirath	bhagirath@gmail.com	3452422343	3423123424	Waratwada, Pune	2002-11-08	F	1

Type 3:- If we are adding data same as of ~~from~~ rows of then ~~column name~~ mentioning column name is not necessary.

Ex.

```
INSERT INTO users
VALUES
(4, "Jen", "Jen@gmail.com", "123456789", "98765432",
"Jodhpur, Rajasthan", "1999-01-10", "M", 1),
(5, "Bhagi", "Bhagi@tech.com", "3452422343", "34231234",
"Varatwada", Pune", "2002-11-08", "F", 0)
;
```

Input

ID 1

ID 2

id	name	email	password	contact	address	dob	gender	status
4	Jen	Jen@	123456	98765	Jodhpur	1999	M	1
5	Bhagi	Bhagi@	3452	34231	Varatwada	2002	F	0

new added

output

output

7 Select .... from database → It gives specific data from database

Type - 1

Eg

```
Select name, email, password from users;
```

(name, email, password: Info asked; users: database; the whole statement: input)

Name	email	Password
John	John@gmail	123456
Jenny	Jenny@gmail	7859527
Bhagirath	bhagirath@gmail	34532798
Bhagi	Bhagi@techg.com	34252444

output

Type - 2

Ex :-

input

```
Select name "Employee name", email, password from
users ;
```

Alias

Name given for email

employee name	email	Password
John	john@gmail	123456
:	:	:
Bhagi	Bhagi@gmail	342324

output

Type 3

Fn

```
Select * from users ;
```

(users: database; * : input)

id	name	email	password	contact ...
1	John	John@gmail	12345678	98765 4321
:	:	:	:	:
5	Bhagi	Bhagi@gmail	345242	3423122

output

Type 4

Ex :-

Select * from users where gender = "M";

input

specific data

id	name	email	password	----	gender
1	John	- - - - - - -			M
2	Jenny	- - - - - -			M
4	Jen	- - - - - - -			M

output

gender only "M" as per demand

Conditional Operators

Symbols	Meaning / Purposes
=	equals
!=	not equals
>=	greater than or equals to
<=	less than or equal to
>	greater than
<	less than

Type 5

Ex :-

```
Select * from users where gender != "m";
```

where → specific data

} input

id	name	- - - - - - -	gender
3	Bhagirath	- - - - - - -	F
5	Bhagi	- - - - - - -	F

output

gender ~~not~~ excluded "m" as per demanded

Type 6

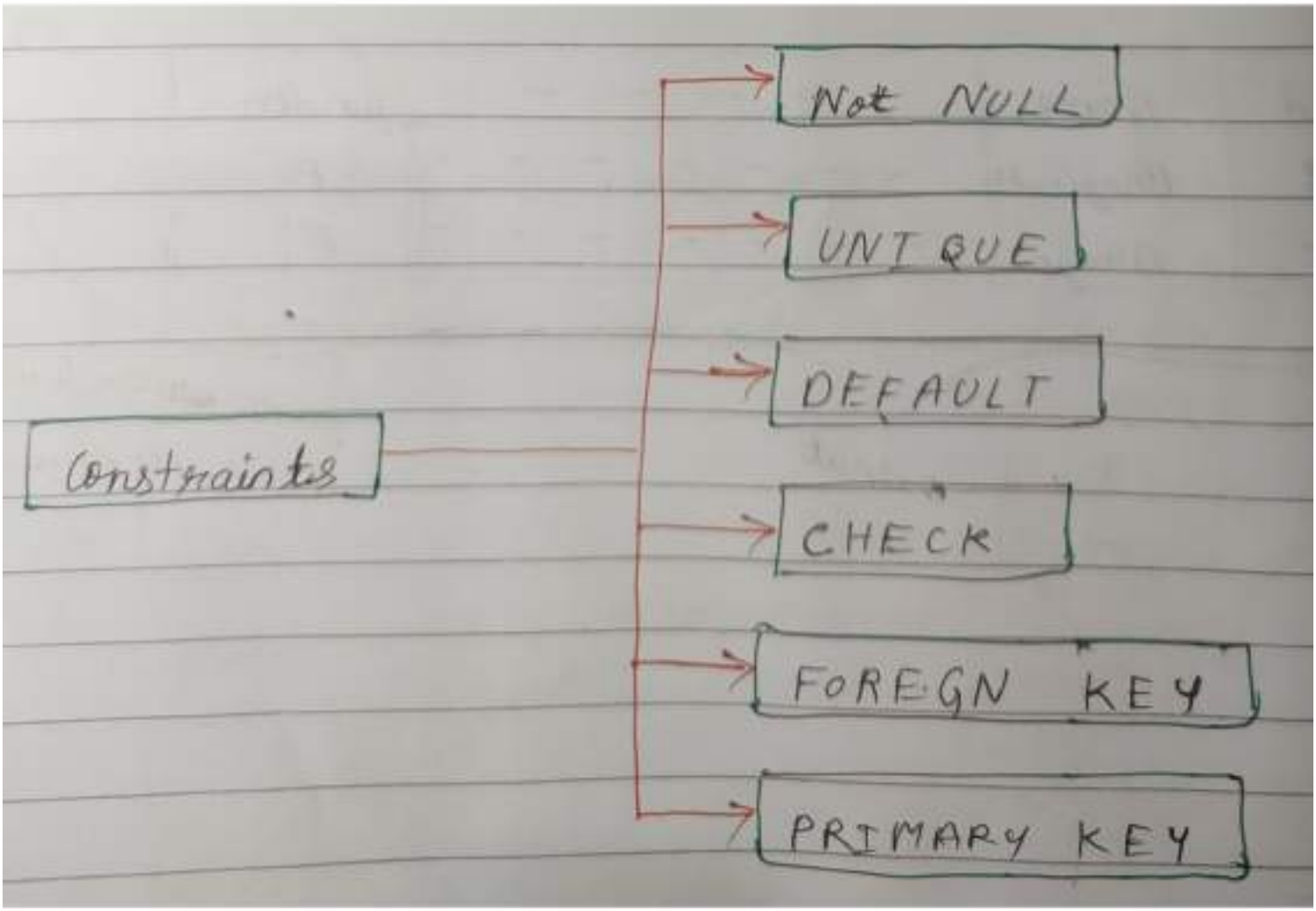
Ex:-

```
select * from users where gender = "o";
```

id	name	email	password	contact	address	dob	gender	sta

* The where clause is used to filter records.

* It is used to extract only those records that fulfill a specified condition


Constraints
Not NULL
UNIQUE
DEFAULT
CHECK
FOREIGN KEY
PRIMARY KEY

Constraints

Not Null → It restricts the ~~data~~ given data i.e data can't be left empty

unique → It restricts the given data i.e It don't allow values to be repeated

default → It modifies the data i.e If data is given ~~use~~ we will consider that and if not then data will automatically considered

Check → It checks wheather a particular condition is satisfied or not.

```
CREATE TABLE Students
(
 id INT NOT NULL unique,

 name varchar(100) not null,

 email varchar(150) not null unique,

 age tiny int check(age >= 18),

 status boolean default 1

);
```

no blank values

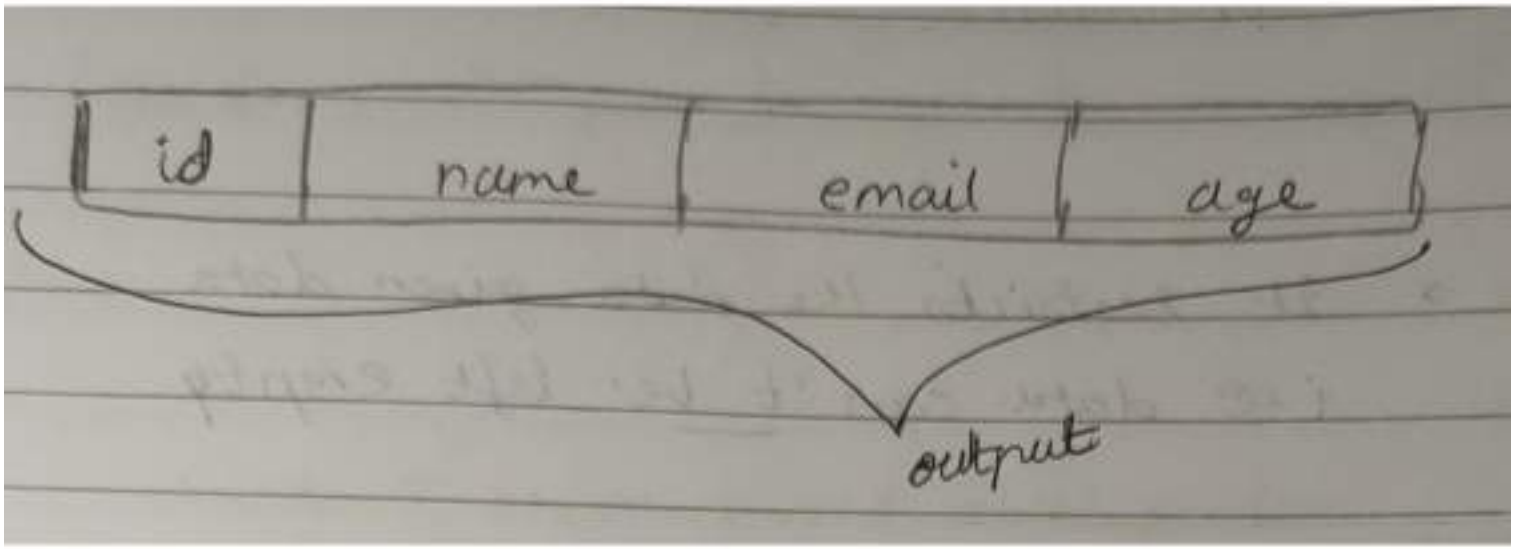
No repeated values

age will check age

age set to greater than 18

0/1

set to 1


id
name
email
age
output

Ex :-

```
INSERT INTO students
(id, name, email, age)
values
(1, "Bhagirath", "Bhagirath@gmail.com", 23)
;
```

Input

id	name	email	age
1	Bhagirath	Bhagirath@gmail.com	23

Output

Attempt - 2

```
INSERT INTO Students
:
:
);
(2, "John", John@gmail.com", 16)
```

Age must be more than 18

Result = Reject

Attempt - 2

```
INSERT INTO Students
:
:
);
(2, "John", John@gmail.com", 16)
```

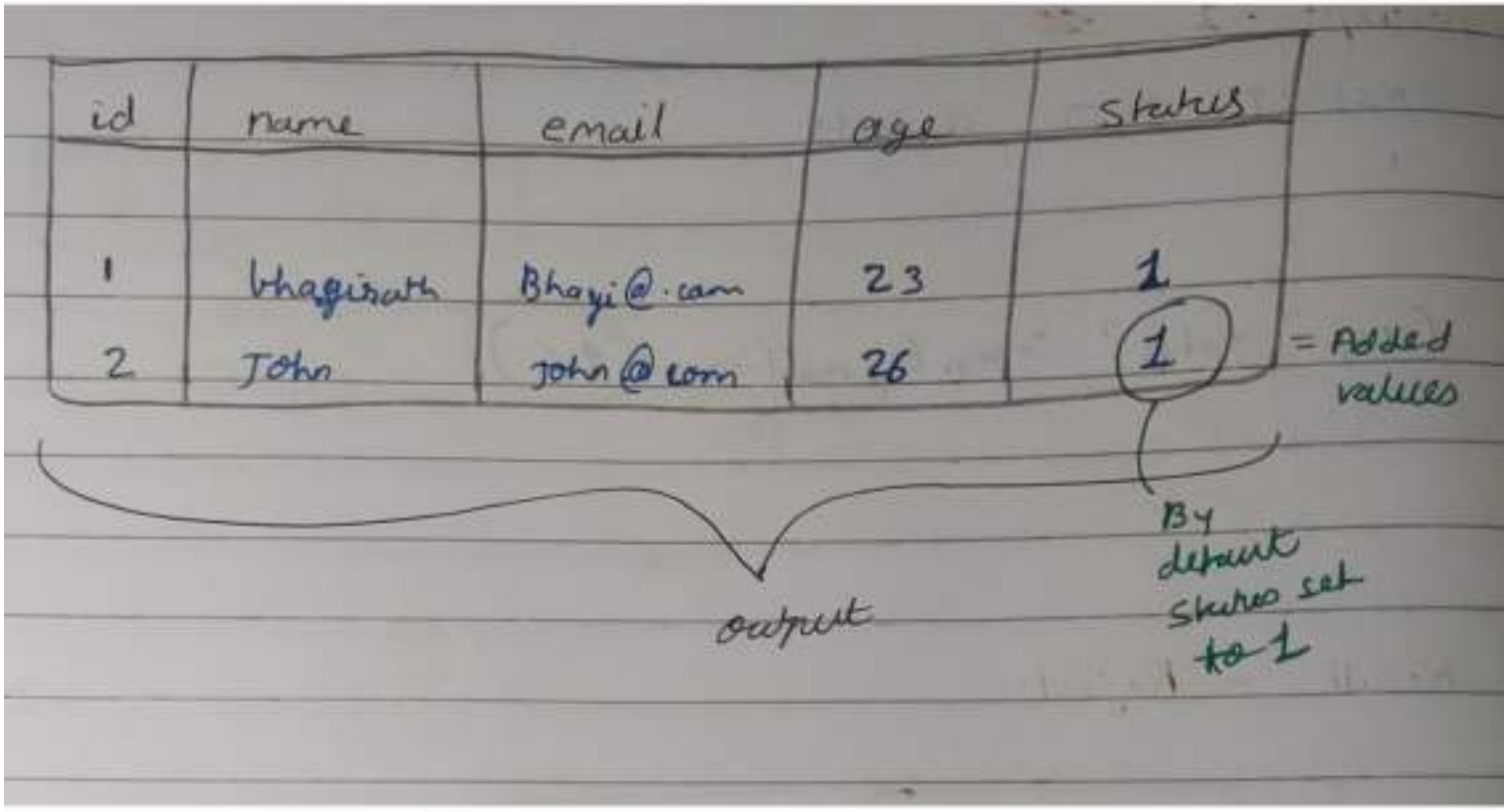
Age must be more than 18

Result = Reject

Attempt - 3

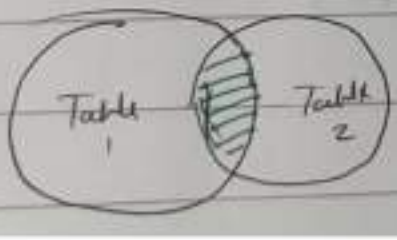
```
INSERT INTO STUDENTS
(id, name, email, age)
values
(2, "John", "John@gmail.com", 26)
;
```

Result = Pass

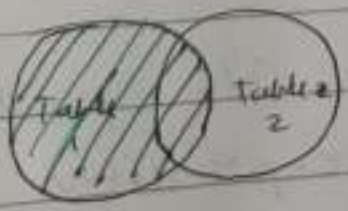

id
name
email
age
status
1
Bhagirath
Bhagi@.com
23
1
2
John
John@com
26
1
= Added values
By default status set to 1
output

SQL JOINs

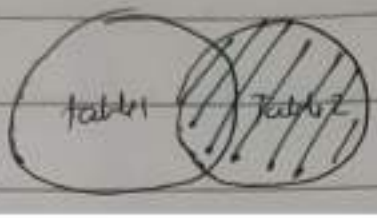
(INNER) JOIN : Returns records that have matching value in both sides



LEFT (OUTER) JOIN :- Returns all records from left table, and matched records from right side



RIGHT (OUTER) JOIN :- Returns all records from right table, and matched records from left table.



FULL (OUTER) JOIN :- Returns all records when there is a match in either left or right

AND operator :- MySQL logical AND operator compares two expressions and returns true if both of expressions are true.

operator

OR ~~operator~~ :- MySQL OR operator compares two expressions and returns TRUE if either of the expression is TRUE

NOT Operator :- MySql NOT operator reverses or negates the input

· ALTER TABLE ⟶ It will add new columns to table.

Before →

id	name	email	age	status
1	bhagirath	bhagirath@	23	1
⋮	⋮	⋮	⋮	⋮
8	kushagra	kushagra@	35	1

} Before code

code {
```
ALTER TABLE students
ADD city varchar(100),      — "city" column will be added
    gender enum("M","O","F"); — "gender" column will be added
```

After →

id	name	email	age	status	city	gender
1	bhagirath	bhagirath@	23	1		
⋮	⋮	⋮	⋮	⋮		
8	kushagra	kushagra@	35	1		

} added columns

·> update table ⟶ It will add new columns values to created columns

Ex :-

```
UPDATE students
Set city = "Jaipur"
where id = 1;
```

"students" table will be updated

city "Jaipur" will be added

id = 1 is location

Before

id	name	email	age	status	city	gender
1	Bhagirath	Bhagirath@	23	1	Null	NULL
⋮	⋮	⋮	⋮	⋮	⋮	⋮
9	Kushagra	Kushagra@	35	1	Null	Null

After

city "Jaipur" will be added

id	name	email	age	status	city	gender
1	Bhag	Bhag@	23	1	Jaipur	NULL
2	John	John@	26	1	NULL	NULL
⋮						
9	Kusha	Kushan@	35	1	NULL	NULL

> Desc table ~~or~~ ⟶ It is used to describe the structure of table, including column names, data types, and any ~~dfp~~ default values, constraints or comments for each column.

~~Feild Type Null Key~~

Ex :-

desc Students ⟵ It describes "Students" values

Feild	Type	Null	Key	Default	Extra
id	int				
name	varchar(100)	No	Pri	Null	
email	varchar(150)	No		Null	
age	tinyint	No	uni	Null	
status	tinyint(1)	Yes		1	
city	varchar(100)	Yes		Null	
gender	enum('M','O','F')	Yes		Null	

·> AND Operation ⟶ It provides values for two conditions at combining at same time.

Ex :- Type 1

id	name	email	age	status	city	gender
1						M
9						M

```
select * from students
where age >= 18 and age <= 25 ;
```

code

age is equal or more than 18

age is equal or less than 18

id	name	email	age	status	city	gender
1			23			
3			19			
4			20			
5			21			
6			25			

Age is between 18 to 25

Type 2 :-

```
Select * from students
where age >= 18 and age <= 25 and city = "Agra"
```

city is only "Agra"

id	name	email	age	status	city	gender
4	Jonny	jonny@	20	1	agra	F
6	ashish	ashish@	25	1	agra	M

Age is 18 to 25

city is only agra

Type 3 :-

Select * from students
where age >= 18 and age <= 25 and city "Agra"
and gender "M"

gender only "M" is selected

id	name	email	age	status	city	gender
6	ashish	ashish@	25	1	agra	M

only gender "M" is selected

·> OR Operation ⟶ It provides values for any conditions.

Ex :- Type 1

```
Select * from students
where city = "agra" or city = "jodhpur";
```

id	name	email	age	status	city	gender
5	Jenny	Jenny@	21	1	Jodhpur	F
7	Pradeep	Pradeep@	28	1	Jodhpur	M
9	Kushagra	Kushagra@	35	1	Jodhpur	M
4	1	1	1	1	agra	1
6	1	1	1	1	agra	1
8	1	1	1	1	agra	1
10	1	1	1	1	agra	1

city "agra" & "jodhpur" are filtered out

• > NOT Operation ⟶ It provides which are not included in content

Ex :- Type 1 (city other than "Agra" are selected)

Where (NOT) city = "Agra"

id	name	Email	age	status	city	gender
1					Jaipur	
2					Jaipur	
3					Dehli	
5					Jodhpur	
7					Jodhpur	
9					Johnpur	

city other than "Agra" are selected

> IN Operator ⟶ It allows you to specify multiple values in a WHERE clause.

- The IN Operator is a shorthand for multiple OR conditions.

```
SELECT * FROM students where age = 20 or age = 21
                    or age = 23;
```

Using OR operation

Type - 1

```
SELECT * FROM students WHERE
age IN (18, 21, 23)
```

input using 'in' operator

using IN operation

id	name	email	age	status	city	gender
1	bhagirath	bhagirath@	23	1	Jaipur	M
3	ankur	ankur@	19	1	Delhi	M
5	Jenny	Jenny@	21	1	Jodhpur	F
10	Ravi	ravi@	23	1	agra	M

Ages of 19, 21, & 23 are selected

Type - 2

```
SELECT * from students WHERE
city IN ("Jodhpur", "Jaipur");
```

using IN operation

id	name	email	age	status	city	gender
1	Bhagirath	Bhagirath@	23	1	Jaipur	M
2	John	John@gmail	26	1	Jaipur	M
5	Jenny	Jenny@gmail	21	1	Jaipur	F
7	Pradeep	pradeep@	28	1	Jaipur	M
9	Kushagra	kushagra@	35	1	Jaipur	M

city "Jodhpur" & "Jaipur" are selected

· 7 Like Operations ⟶ It is used to search for a specified pattern in a column.

Ex :-

id	name	email	password	contact	address - - -	status
1	John	John@	123456			
5	Bhagi	Bhagi@	34529			

```
SELECT * FROM users
where name like "J%";
```

Ed name is selected

id	name	email	password	contact	address	dob	g	status
1	John	John@	- - - - - -				M	1
2	Jenny	Jenny@	- - - - - -				M	1
4	Jen	Jen@	- - - - - -				M	1

Names starting with "J" are selected

∴ The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Like Operators	Description
Like 'a%'	Starts with "a"
Like '%a'	End with "a"
~~Like '%or%'~~	
Like '%or%'	Have 'or' in any position
Like '_r%'	Have "r" in the second position
Like 'a_%'	Starts with "a" and are at least 2 characters in length
Like 'a__%'	Starts with "a" and are at least 3 characters in length
Like 'a%o'	starts with "a" and ends with "a"

Ex Type 1

```
SELECT * from students
where name like "%a";
```

} input

id	name	email	age	city	gender	status
8	Riya	Riya@	30	agra	F	1
19	kushagra	kushagra@	35	[illegible]	M	1

} output

name ends with 'a'

Type 2

```
SELECT * from students
where name like "a%";
```

input

id	name	email	age	city	gender	status
3	ankur	ankur@	19	Delhi	M	1
6	ashish	ashish@	25	agra	M	1

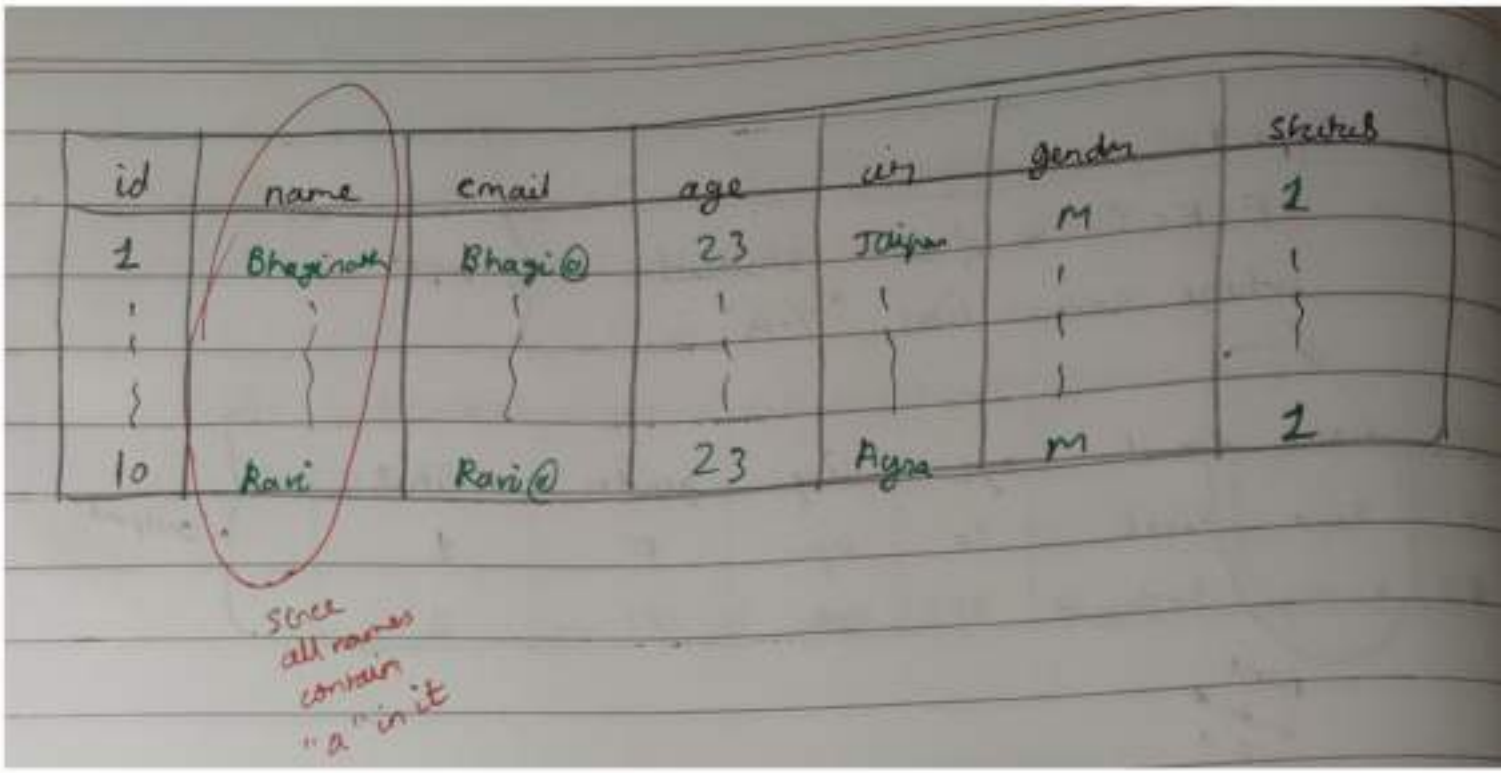
output

name starts with "a"

Type 3

```
SELECT * from students
where name like "%a%";
```

input


id
name
email
age
city
gender
Student
1
Bhagirath
Bhagi@
23
Jaipur
M
1
10
Ravi
Ravi@
23
Agra
M
1
Store all names contain "a" in it

Type 4

SELECT * FROM students
where name Like "a_____";

After "a" 4 places are selected

id	name	email	age	city	gender	status
3	ankur	ankur@gmail.com	19	Delhi	M	1

Here name that contains "a" at start and has 5 words after it

Type 5

```
SELECT * FROM students
WHERE name LIKE "a____%";
```

after "a"
4 places out
selected then
any word

id	name	email	age	status	city	gender
3	ankur	ankur@	19	1	delhi	M
6	aohish	aohish@	25	1	agra	M

here name that contain "a" in first letter and 4 or more letters are filtered out

Type 6

```
SELECT * FROM Students
where name like "%_a_%";
```

Here "a" is between names are been filtered out

id	name	email	age	city	gender	status
1	Bhagirath	Bhagi@	23	Jaipur	M	1
7	Pradeep	Pradeep@	28	Jodhpur	M	1
9	Kushagra	kusha@	35	Jodhpur	M	1
10	ravi	ravi@	23	Jodhpur	M	1

One word before "a", one word after "a" and rest any letter word containing any letter are filtered out

.> BETWEEN OPERATOR → It is used to check whether a value lies within a specific range.

Ex :- Type 1

```
SELECT * from students
where age between 20 and 25 ;
```

between operator will tell the students whose age is between 20 to 25

id	name	email	age	status	city	gender
1	bhavinath	bhavinath@	23	1	Jaipur	M
4	Jonny	JONAY@	20	1	agra	F
5	Jenny	Jeny@	21	1	dobhpur	F
6	ashish	ashish@	25	1	agra	M
10	ravi	ravi@	23	1	agra	M

Ages between 20 to 25 has been filtered out

Type 2

```
SELECT * from students
where age NOT between 20 and 25;
```

Here students who's age is not between 20 to 25 are selected

id	name	email	age	status	city	gender
2	John	John@	26	1	1	M
3	ankur	ankur@	19	1	1	M
7	pradeep	pradeep@	28	1	1	M
8	riya	riya@	30	1	1	F
9	kushagra	kushag@	35	1	1	M

Here age
except 26 &
25 are been
filtered out

7. ORDER BY clause → It is used in SQL to sort the result set of a query based on one or ~~two~~ more columns. It allows you to specify the order in which the rows should appear in query result.

Type 1

```
SELECT * from students
order by name ASC;
```

input

name → column name

ASC → list will be arranged in ascending order

~~id name email age~~

id	name	email	age	status	city	gender
3	ankur					M
6	ashish					M
7	pradeep					M
10	ravi					M
8	riya					F

output (list is arranged in alphabatical order

Type 2

```
SELECT * FROM Students
ORDER By name DESC;
```

(Students: table name; name: column name; DESC: descending order; input)

id	name	email	age	status	city	gender
8	riya	riya@	30	1	agra	F
10	ravi	ravi@	23	1	agra	M
⋮	⋮	⋮	⋮	⋮	⋮	⋮
6	aakash	aakash@	25	1	agra	M
3	ankur	ankur@	19	1	delhi	M

output (list of data we get in descending order)

Type 3

```
SELECT * FROM students
ORDER by age ASC
```

} input

id	name	email	age	status	city	gender
3	ankur		19			
4	Jonny		20			
8	riya		30			
9	kusha		35			

output (list is filtered out by age in ascending order)

> DISTINCT ⟶ It shows unique values of table

Type - 1

Eg :- select distinct city from students ; } input

city → column city is here used

city
Jaipur
delhi
agra
Jodhpur

} output

Type 2

Eg :- SELECT DISTINCT age from students ;

input

"age" column is selected

age
23
26
19
20
21
25
28
30
35

output (list contains all unique values of age column)

Type 3

Fn :- SELECT DISTINCT
age from students
order by age ASC

} input

ASC → ascending order

age
19
20
⋮
30
35

} output (list contains all unique values in ascending order)

Type 4

Ex :-

```
SELECT DISTINCT city
FROM students
ORDER By city ASC
```

} input

city
ajna
delhi
Jaipur
Jodhpur

} output (list contains all cities unique name by ascending order)

→ NULL ⟶ It refers to values which are empty i.e no data is inserted into it

Type 1

Ex :-

input

```
insert into Students
values
(11, "shaili", "shaili@gmail.com", 24, 1,
"Jodhpur", NULL );
```

NULL → no value

id	name	email	age	status	city	gender
1	bhagirath	bhagirath@	23	1	Jaipur	m
⋮	⋮	⋮	⋮	⋮	⋮	⋮
11	shaili	shaili@	24	1	Jodhpur	NULL

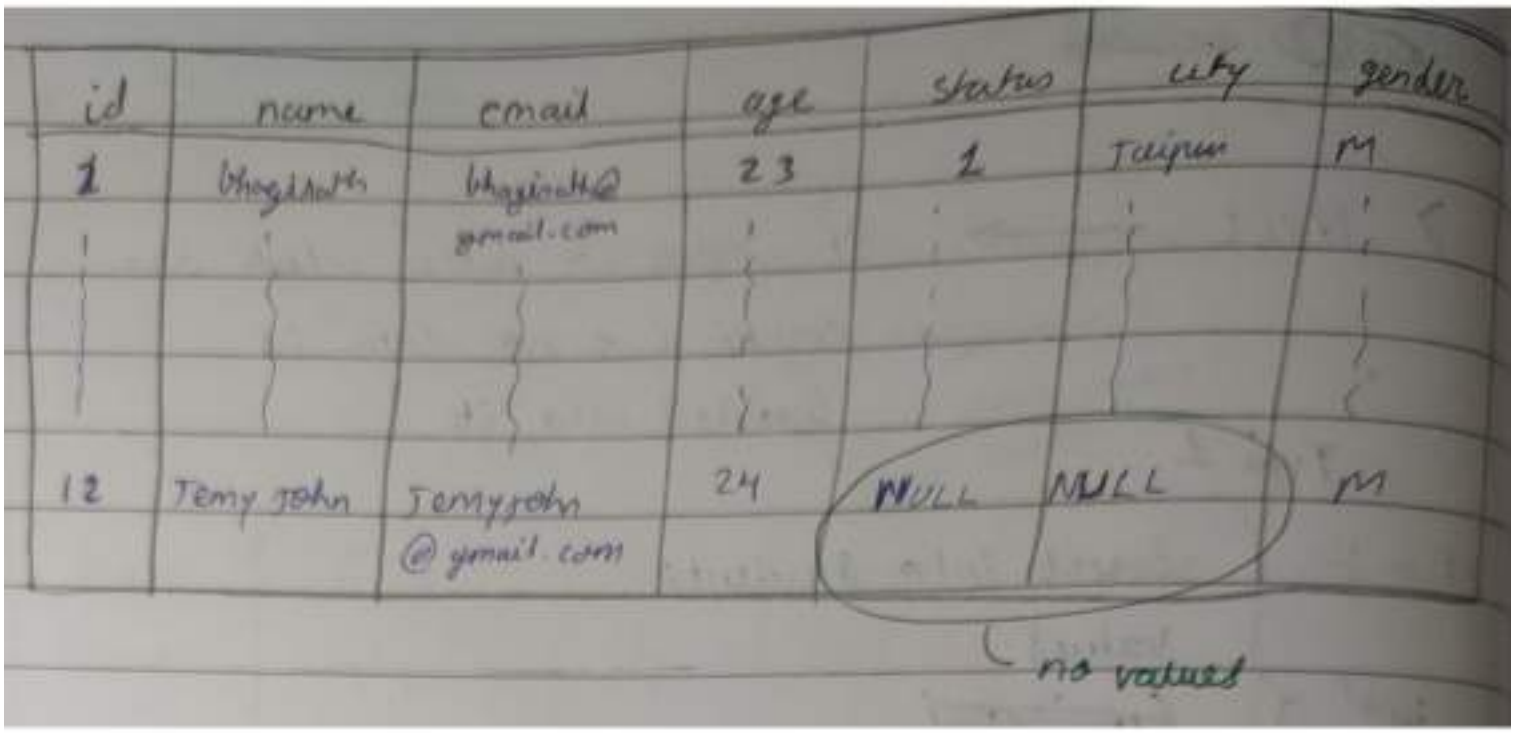
null value

Output

Type 2

Ex :- insert into Students
values
(12, "Jenny John", "Jennyjohn@gmail.com",
24, NULL, NULL, 1);

no value


id
name
email
age
status
city
gender
1
Bhagirath
bhagirath@gmail.com
23
1
Jaipur
M
12
Jemy John
Jemyjohn@gmail.com
24
NULL
NULL
M
no values

Type 3

Ex :- ~~insert~~

```
Select * from students
where gender is NULL ;
```

} input

id	name	email	age	status	city	gender
11	Shaili	shail@gmail.com	24	1	Jodhpur	NULL
12	Jenny John	Jennyjohn@gmail.com	24	1	NULL	NULL

gender is "NULL"

output

• NO NULL ⟶ It refers to condition where we can't leave empty data i.e we need to put some data

Type 4

Ex :-

```
Select * from students
where gender is not null ;
```

input

gender must have some values

id	name	email	age	status	city	gender
1	bhagirath	bhagirath@gmail.com	23	1	Jaipur	M
⋮	⋮	⋮	⋮	⋮	⋮	⋮
10	ravi	ravi@gmail	23	1	agra	M

output

all columns have some values

Type 5

Ex:-

```
Select * from students
where gender is NULL and
city is NULL ;
```

city & gender are "NULL"

id	name	email	age	status	city	gender
12	Jenny John	jennyjohn@gmail.com	24	1	NULL	NULL

· > LIMIT ⟶ It sets the limit for any command.

∴ It starts from '0' then 1, 2, 3 - - - - and so on.

Type 1

Ex :-

```
Select * from
students limit 5 ;
```

} input

limit is set to 5

id	name	email	age	state	city	gender
1	Bhagirath	bhagirath@gmail.com	23	1	Jaipur	M
⋮	⋮	⋮	⋮	⋮	⋮	⋮
5	Jenay	Jenny@gmail.com	21	1	Jodhpur	F

output

Type 2

Eg :-

input

```
SELECT * from students
ORDER BY name
DESC LIMIT 5 ;
```

column name

descending

only upto 5

id	name	email	age	status	city	gender
11	Shaili	—	—	—	—	NULL
9	Riya	—	—	—	—	F
10	Rani	—	—	—	—	M
7	pradeep	—	—	—	—	M
9	keshavgarg	—	—	—	—	M

output All values are sorted in descending order based on name

Type 3

Eg Fn:- SELECT * from Students
ORDER By name desc ;

input

id	name	gender
11	shalki	NULL
8	riya	F
⋮	⋮	⋮
6	ashish	M
3	ankur	M

output

Type 4

Ex 7 SELECT * FROM students
ORDER BY age DESC ;

input

age → column name

id	name	email	age	status	city	gender
9	kushagra		35			
8	Riya		30			
4	tony		20			
3	ankur		19			

age is sorted in descending order

output

Type 5

Ex - select * from students

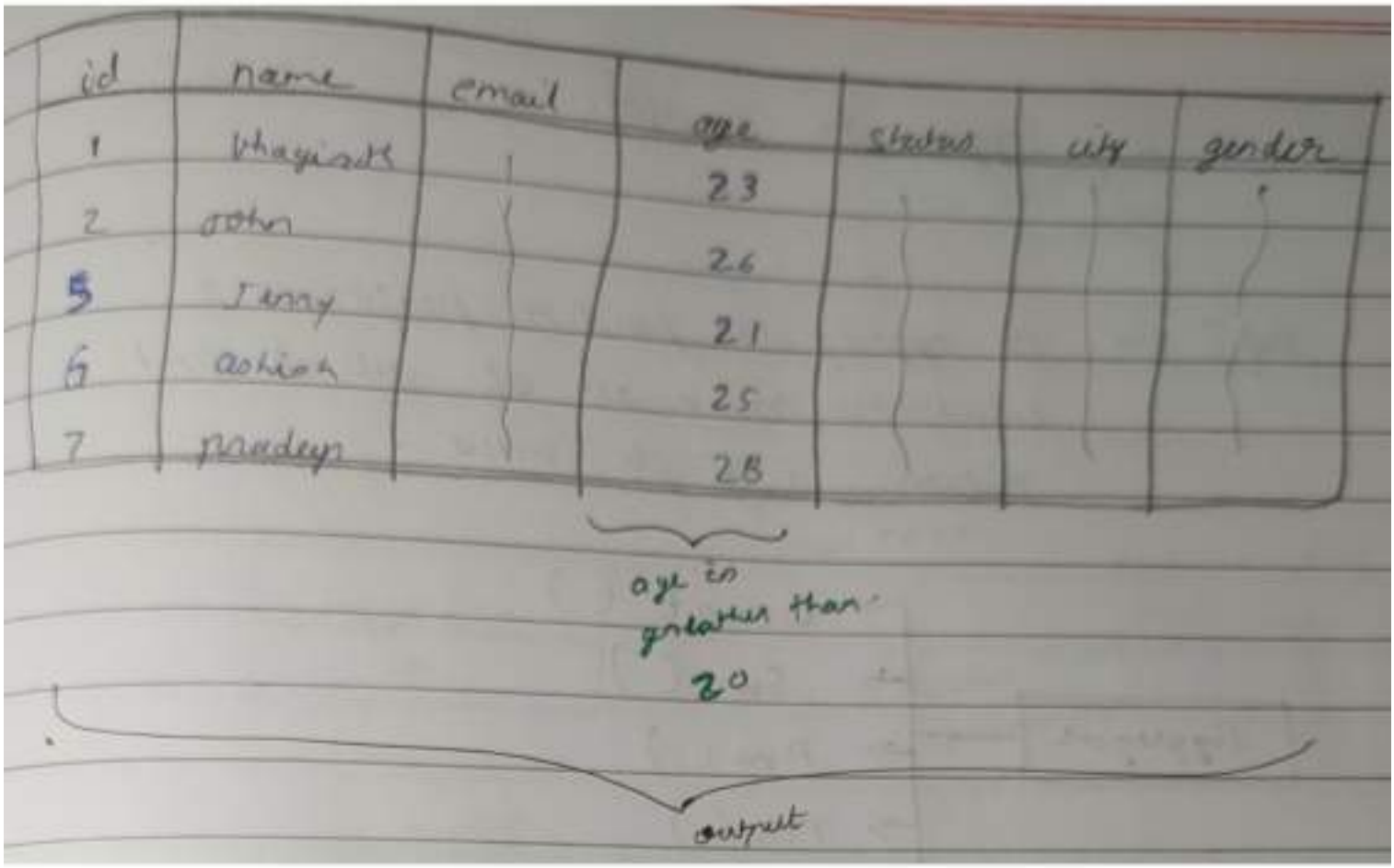
where age > 20 limit 5 ; — limit is set to 5

(age is greater than 20)

~~id name email age student city gender~~

* ~~[illegible]~~

input


id
name
email
age
status
city
gender
1
Khagendra
23
2
John
26
5
Binay
21
6
Ashish
25
7
Pradeep
28
age is greater than 20
output

> OFFSET ⟶ It is use to specify from where to start returning data.

Ex - SELECT * FROM students ← Table name
LIMIT 5 OFFSET 2

LIMIT 5 → limit is set to 5

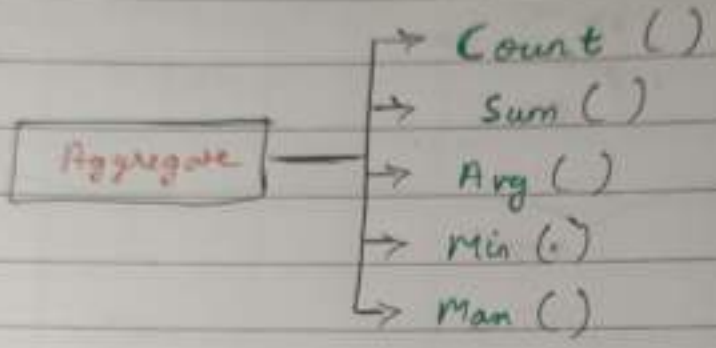
OFFSET 2 → first 2 values are skipped

id	name	email	age	Status	city	gender
3	ankush	ankush@	19	1	Delhi	M
4	gonay	jonny@	20	1	agra	F
5	Jenny	Jenny@	21	1	jodhpur	F
6	ashish	ashish@	25	1	Agra	M
7	pradeep	pradeep@	28	1	Jodhpur	M

values from 3 to 7 has fetched out

Aggregate function

Def^n :- An aggregate function performs a calculation on a set of values and returns a single value.



Count () :- returns the number of rows in a table or a selected set of rows.

Sum () :- returns the sum of a set of values.

Avg () :- returns the average of a set of values.

Min () :- returns the minimum value in a set

Max () :- returns the maximum value in a set

Type 1

Ex :- SELECT count(id)
FROM students
WHERE fees > 5000;

aggregate function

input

fees greater than 5000

Kushagra - 8000
ram - 6000

count(id)
2

2 students has fees more than 5000

output

Type 2

Eg :- input

```
SELECT sum(fees)
FROM students ;
```

aggregate function

5000 + 4000 + 5000
+ 8000 + 6000
= 28000

output

Count(id)
28000

→ Total no. of fees of all students

Type 3

Ex - input
```
SELECT avg (fees)
FROM students ;
```
aggregate function

$\frac{5000+4000+5000+3000+8000}{5}$

output

Count (id)
5600.00

— average fees of 5 students

$\frac{28000}{5} = 5600$

Type 4

Ex :- SELECT avg (fees) as 'avgfees' FROM students ;

aggregate function

alias name

input

output

avgfees
5600

new column name

Type 5

input

```
Fn -  SELECT  min(fees)
      FROM students ;
```

aggregate function

Table name

output

min(fees)
4000

minimum fees

Type 6

input

```
Ex :- SELECT max (fees)
      FROM Students;
```

aggregate function

table name

output

max (fees)
8000

——— maximum fees

Type 7

Ex

```
SELECT max(age)
FROM students;
```

aggregate function

max(age)
35

• > UPDATE statement ⟶ It is used to update data of the table

Type 1

Ex :-

```
update students
set age = 20          ---- age will be updated to 20
where id = 2 ;        ---- id = 2 is John so his age will be changed to 20
```

id	name	email	age	status	city	gender	fees
1	Bhagirath	Bhagirath@gmail.com	23	1	Jaipur	M	5000
2	John	John@gmail.com	20	1	Jaipur	M	NULL
12	Jonny John	Jonny John@gmail.com	24	1	NULL	NULL	NULL

Type 2

Ex:-

input

```
UPDATE Students
Set age = 35      ---- we are setting age = 35
where id = 5;     ---- id = 5 's age will
                       be modified
```

id	name	email	age	status	city	gender	fees
1	Bhagirath	Bhagirath @gmail.com	23	1	Jaipur	M	5000
5	Jerry	Jerry@ gmail.com	35	1	Jodhpur	F	NULL
12	Jerry John	Jerry John @gmail.com	24	1	NULL	NULL	NULL

output

- DELETE ⟶ It lets us erase data from database

Type 1

Eg delete from students ← Table name
where id = 4 ; ← id = 4 will be erased

id	name	email	age	status	city	gender	fees
1	Bhagirath	bhagirath@gmail.com	23	1	Jaipur	M	5000
3	ankur [redacted]	ankur@gmail.com	19	1	Delhi	M	Null
5	Jenny	Jenny@gmail.com	35 [redacted]	1	Jodhpur	F	NULL
12	Jenny John	Jenny.john@gmail.com	24	1	NULL	NULL	NULL

id = 4 is erased

Type 2

Eg :- delete from students
where id IN (5,6); — ids 5 & 6 will be erased

id	name	email	age	status	city	gender	fees
1	bhagirath	bhagirath@gmail.com	23	1	Jaipur	M	5000
3	john	john@gmail.com	19	1	Delhi	M	NULL
7	pradeep	pradeep@gmail.com	20	1	Jodhpur	M	4000
12	Jennyjohn	Jennyjohn@gmail.com	24	1	NULL	NULL	NULL

(ids 5 & 6 are deleted)

∴

alter table students
modify id int
NOT NULL auto_increment ;

NOT NULL: column will not remain empty

auto_increment: column will take automatic +1 values

Type 3

```
insert into students
(name, email, age, status, city, gender, fees)
values
("aman", "aman@gmail.com", 23, 1,
"NULL", "M", 5000)
;
```

we are not providing "id"

id	name	email	age	status	city	gender	fees
1	Bhagirath	bhagirath@gmail.com	23	1	Jaipur	M	5000
12	Jerry John	jerryjohn@gmail.com	24	1	NULL	NULL	NULL
13	aman	aman@gmail.com	23	1	NULL	M	5000

id is automatically created

. > Primary key → Primary key always contains unique data and can't be null. There must be a single primary key

.> Foreign key ⟶ The foreign key is used to link two tables. A foreign key is one table (child table) is used to point PRIMARY KEY in another table (parent table).

Ex :- ~~Create table classes~~

~~Create~~

Create table cities

(

cid ~~cid~~ int auto_increment — "cid"

name varchar(100), — "name"

primary key (cid) — "cid" is primary key

);

cid	name

output

~~insert into cities~~

Field	Type	NULL	Key	Default	Extra
cid	int	NO	PRI	NULL	auto_increment
name	varchar(100)	YES		NULL	

input

```
insert into cities
(name)
values
("agra"),
("bhopal"),
("delhi"),
("jaipur"),
("raj kot");
```

This 5 data will be created

output

cid	name
1	agra
2	bhopal
3	delhi
4	jaipur
5	raj kot

Type 1

```
Create table students1 ( ——— "students1"
(
id int not null unique auto_increment, ——id
name varchar(100) not null, ——— name
email varchar(100) not null unique, ——email
city_id int null, ——— city_id
primary key (id), ———"id" is primary key
foreign key (city_id) references cities (cid)
);
```

"city_id" is foreign key

"cid" is reference of cities

Feild	Type	NULL	Key	Default	Extra
id	int	No	PRI	NULL	auto_increment
name	varchar (100)	No		NULL	
Email	varchar (100)	No	UNI	NULL	
city_id	int	YES	MUL	NULL	

INSERT INTO students
(name, email, city_id)
values
('Bhagirath', 'bhagirath@wscubetech.com', 1);

Id = 1
is created

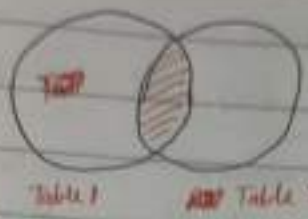
id	name	email	city_id
1	bhagirath	bhagirath@wscubetech.com	1

output

INSERT INTO students
(name, email, city_id)
values
('Ravi', 'Ravi@wscubetech.com', 10);

∴ Error ——— as foreign key is assigned to table cities

1) INNER JOINT → A type of join



Ex -

input

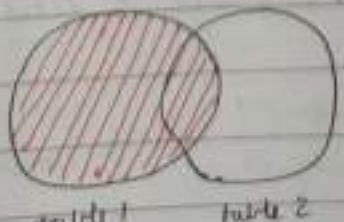
```
SELECT * FROM Students1 "Students1"
INNER JOIN CITIES
ON Students1.city_id = cities.cid;
```

table = Students1 table = cities
column = city_id column = cid

id	name	email	city_id	city	name
1	bhagirath	bhagirath@gmail.com	1	1	agra

output (tables like students1 & cities are showing data in a joint manner

2) LEFT JOINT →



Ex :-

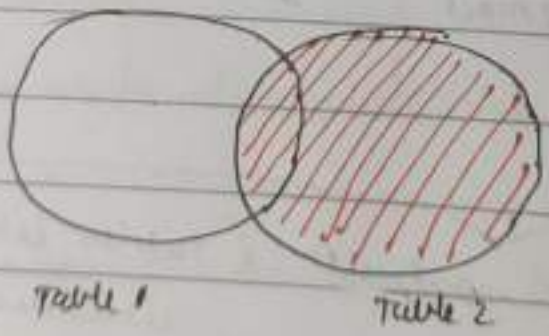
Table name

```
SELECT * FROM students1
    left join cities
    on students1.city-id = cities.cid;
```

table = students1 | table = cities
column = city-id | column = cid

id	name	email	city-id	cid	name
1	bhagirath	bhagirath@gmail.com	1	1	agra

> RIGHT JOINT ⟶



Ex

```
SELECT * FROM Students
RIGHT Join cities
on Students1.city-id = cities.cid ;
```

table = students1
column = city-id

table = cities
column = cid

id	name	email	city_id
1	bhagirath	bhagirath@gmail.com	1
NULL	NULL	NULL	NULL
⋮	⋮	⋮	⋮

cid	name
1	agra
2	bhopal
3	delhi
4	Jaipur
5	Rajkot

> CROSS JOINT ⟶

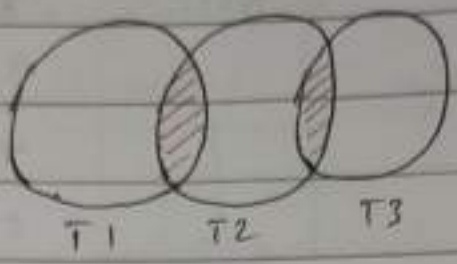
Table 1 Table 2

Ex

```
SELECT * from students    ← Table "students"
CROSS JOIN cities;        ← Table "cities"
```

id	name	email	age	status	city	gender	fees
1	bhagirath	bhagirath@	23	1	Jaipur		
⋮	⋮	⋮	⋮	⋮	⋮		
13	aman	aman@	23	1	NULL		

7 Multiple Table joins




```
SELECT * FROM cities
Join students
ON cities.city_id = students.city_id
Join students1
ON students.student_id =
students1.student_id ;
```

"cities" — "students" — columns — "students1" — column

·> SUB - QUERY :⟶ It is nested inside SELECT, INSERT, UPDATE or DELETE statement or inside other subquery.

Ex :- select cid from students where name = "agra"; } input

cid
1

} output

Ex :- Select name from students
where id =
(select id from students where
city = "delhi");

input

name
ankur

output

Ex :- Select name from students
where exists (Select id *
from cities where city = "delhi");

input

o/p is students will be shown

name
ankur

output

Ex :- Select name from students
where not exists
(SELECT id from cities
where city = "delhi");

input

name
bhagirathi
John
pradeep
raja
:
aman

output

. > GROUP BY ⟶ It is used to group rows that have same values in specified columns.

Ex :- SELECT * from students group by (id) ; } input

clause

id	name	fees
1	bhagirath	15000
:	:	:
13	aman	5000

Ex :- Select id , count (id)
from students group by (id) ; } input

clause

clause

id	count (id)
1	1
2	1
3	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1

output

Ex :-

```
select id, cities name, count(id)
from students
inner join cities
on students.id = cities.cid
group by (id);
```

input

id	name	count (id)
1	agra	1
2	bhopal	1
3	delhi	1

output

Ex

```
Select id, cities.name, count(id)
as total from students
inner join cities
on students.id = cities.cid
group by (id)
having count(id) >= 2;
```

input

id	name	count (id)
1	agra	1
2	bhopal	1
3	delhi	1

output