17/02/2023

Notes:-

What is Database:-
A database is a systematic collection of data. They support electronic storage and manipulation of data. Databases make data management easy.

How to create Database:-
Syntax- CREATE DATABASE DATABASE_NAME;

How to see Database:-
Syntax:- **SHOW DATABASE;**

How to use Database:-
Syntax:- **USE DATABASE_NAME;**

How to delete Database:-
Syntax:- **DROP DATABASE DATABASE_NAME;**

What is SQL:-
-Structured Query Language
-SQL lets you access and manipulate database

Database Table:- A database most often contains one or more tables. Each table is identified by a name. Tables contain records with data.

Entity:- All the relative information collected in the table is known as entity.

Rows can be also known as Tuple / Record

Column can also known as Field / Attribute / Feature

Create Table:-The CREATE TABLE statement is used to create a new table in a database.

Syntax:-
```
CREATE TABLE table_name (
        column1 datatype,
        column2 datatype,
        column3 datatype,
  ....
);
```

- The column parameters specify the names of the columns of the table.
-The datatype parameter specifies the type of data the column can hold.

Insert INTO :- The INSERT INTO statement is used to insert new records in a table.

Syntax:-
```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

-If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

Syntax for that is:-
```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

UPDATE :- The UPDATE statement is used to modify the existing records in a table.

Syntax:-
```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

PRIMARY KEY :-

- It should be unique
- Table can only have one primary key
- Cannot contain NULL values

Syntax :-

```
CREATE TABLE table_name (
        column1 datatype,
        column2 datatype,
        column3 datatype,
        primary key (column1)
```

How to create a PRIMARY KEY after creating a table ?
We can use ALTER

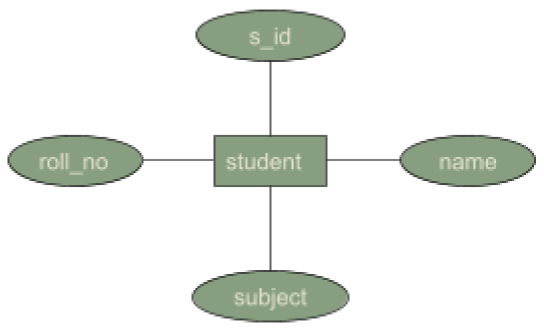
Syntax:-

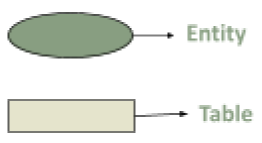
```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

ER Model:- Entity Relationship Modelling

Overview of how we represent ER model :-

In Detail ER Model





This is the student table representing the ER model

s_id	name	subject	roll_no

SQL COMMANDS:-

1. DDL:- Data Definition Language
    - CREATE
    - DROP
    - ALTER
    - TRUNCATE
2. DML:- Data Manipulation Language
    - INSERT
    - UPDATE
    - DELETE
3. DCL:- Data Control Language
    - GRANT
    - REVOKE

CASE STUDY

```
# create, show, use  database
create database aneri;
                                          show databases;
                                          use aneri;



# create table student
create table student(
                                           id int,
     username varchar(100));

# insert the records into student table
                                          insert into student values (1, 'aneri');
                                          insert into student (username) values ('Mia');
                                          insert into student values(4, 'vera'), (5, ' mars');

# to select all the values from student table we use * sign

select * from student;

# create table person
                                          create table person(
                                          pid int,
                                          pfirstname varchar(100),
                                          plastname varchar(100),
                                          address varchar(100),
                                          country varchar(30),
                                          mobileno int,
                                          dob date
                                          );

# insert values into person table

insert into person values (1, 'abc', 'def', 'hjdch', 'Germany', 23564464, '1996-02-22');
insert into person values (2, 'cbc', 'vzf', 'mnop', 'India', 85656564, '1998-05-29');
insert into person values (3, 'dbc', 'eef', 'qrstv', 'Italy', 23964464, '1995-09-12');

# select all the records from the person table

select * from person;




# create table employee

                                          create table employee(
                                          eid int,
                                          efirstname varchar(50),
                                          elastname varchar(50),
                                          edepartment varchar(100),
                                          egender varchar(30),
                                          edob date,
                                          esalary int,
                                          ejoiningdate date,
                                          emobileno int,
                                          ecountry varchar(200),
                                          eaddress varchar(200),
                                          emailid varchar(100));

# describe columns we can use
```

describe employee;

# update query

update student set id=2 where username='Mia';
update student set username= 'mars' where id=4;

# create table using primary key

```
create table student1 (
s_id int,
s_name varchar(100),
primary key(s_id) );
```

# insert values into student1 table

```
insert into student1 values (1, 'ann');
insert into student1 values (1, 'dse');
```

# add primary key after creating table using ALTER

alter table student add primary key (id);

# how to assign one character for gender using data type char(1)

```
create table student12 (
s_id int,
s_name varchar(100),
gender char(1) character set ascii,
primary key(s_id) );
```

# insert values into student12 table

```
insert into student12 values (1, 'ann', 'f');
insert into student12 values (1, 'ann', 'ff');
insert into student12 values (2, 'cef', 'm');
```

# select all the values from student12 table

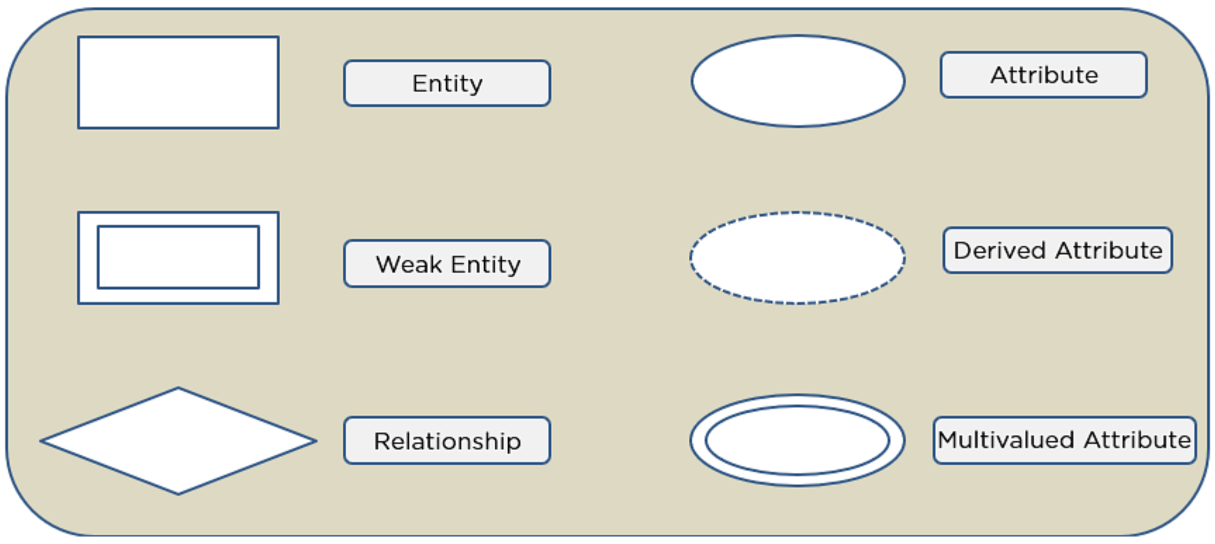
select * from student12;

20/02/2023

ER MODEL :- An Entity-Relationship Model represents the structure of the database with the help of a diagram. ER Modelling is a systematic process to design a database as it would require you to analyze all data requirements before implementing your database.

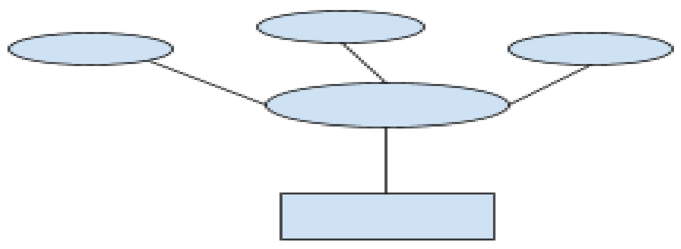
Symbols Used in ER Diagrams

- **Rectangles**: This Entity Relationship Diagram symbol represents entity types
- **Ellipses**: This symbol represents attributes
- **Diamonds**: This symbol represents relationship types
- **Lines**: It links attributes to entity types and entity types with other relationship types
- **Primary key**: Here, it underlines the attributes
- **Double Ellipses**: Represents multi-valued attributes



Weak Entity :- An entity that makes reliance over another entity is called a weak entity.

Composite Attribute :- An attribute that is composed of several other attributes is known as a composite attribute.



Multivalued Attribute:- Some attributes can possess over one value, those attributes are called multivalued attributes.
Example:- phone number, email there can be more than one phone number and email.

Derived Attribute:- A column which can be prepared by another column.

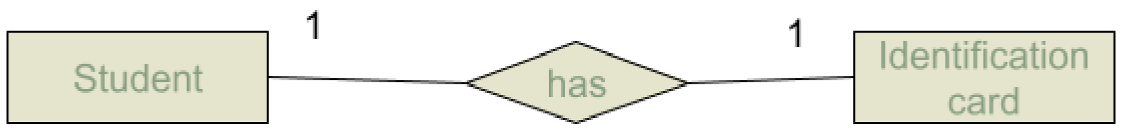
- An attribute that can be derived from other attributes of the entity is known as a derived attribute.

Relationship :- It depicts the relationship between two entities.
- The diamond shapes showcases a relationship in the ER diagram.

Types of Relationship:-

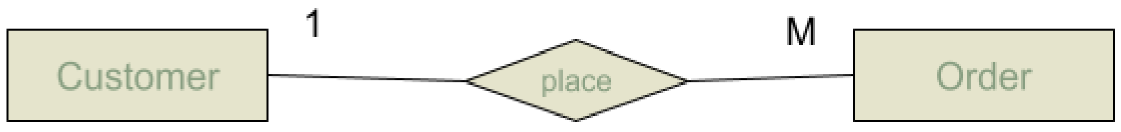
1 .One-to-One Relationship :- When a single element of an entity is associated with a single element of another entity, it is called a one-to-one relationship.

Example:- A student has only one identification card, and a particular identification card is only given to one student.



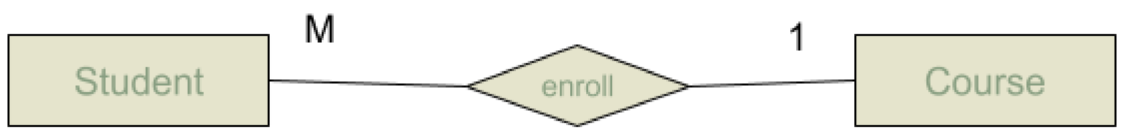
2. One-to-Many Relationship :- When a single element of an entity is associated with more than one element of another entity, it is called a one-to-many relationship.

Example:-A customer can place many orders, but an order cannot be placed by many customers.



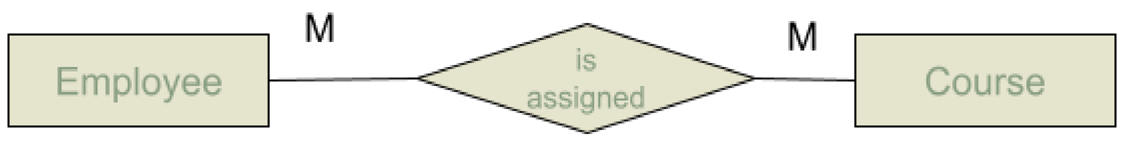
3. Many-to-one Relationship :- When more than one element of an entity is related to a single element of another entity, then it is called a many-to-one relationship.

Example:- A student has an option to enroll in a single course but a course can have many students.



4. Many-to-Many Relationship :- When more than one element of an entity is associated with more than one element of another entity, this is called a many-to-many relationship.

Example:- An Employee can be assigned to many projects and a project can have many employees.



SQL KEYS

Foreign key :-
- A foreign key is key used to link two tables
- A Foreign key is a field in one table that refers to the PRIMARY KEY in another table.

Candidate Key :-
- Candidate key is a combination of multiple columns that help in identifying each row distinctly.
  Example:- id - PK, name, address, passport_no, gov_id
- So in the above described column names we can say that id, passport_no and gov_id are candidate keys, because from any of the three columns we can get a record of any person.

Composite Key:- Combination of all the keys are known as Composite Key.

Researched Topic In Detail SQL KEYS

SQL KEYS are single or multiple attributes used to get data from the table according to the requirement or condition. They can also be used to set up relationships amongst various tables. There are several types of

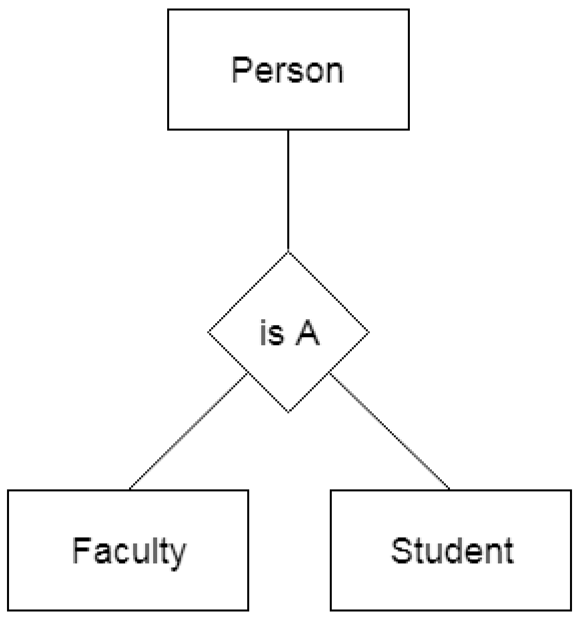
Types of Keys

1. **SUPER KEY:-** A SUPER KEY is a combination of one or multiple columns in a table in the database, which help in the unique identification of each row in the table. It is a group of one or several keys.
2. **CANDIDATE KEY :-** CANDIDATE KEY is also a set of columns or attributes that help identify each row in the table separately. A CANDIDATE KEY can be defined as a SUPER KEY having no matching attributes. It can be demonstrated as a sub-set of SUPER KEY. Several candidate keys can be there in a table.
3. **PRIMARY KEY :-** PRIMARY KEY is an attribute or a group of attributes that help in identifying individual rows distinctly. There cannot be the exact value of the PRIMARY KEY more than once in the table. A PRIMARY KEY can be expressed as a sub-set of a CANDIDATE KEY. There cannot be Multiple PRIMARY KEYS in a table.
4. **FOREIGN KEY :-** A FOREIGN KEY in a table is an attribute that establishes a relationship between two tables. A FOREIGN KEY of one table references the PRIMARY KEY of another table, establishing the relation between the two tables. A FOREIGN KEY can accept multiple null and duplicate values.
5. **ALTERNATE KEY :-** ALTERNATE KEY helps in identifying the records in the table distinctly. There can be several columns in a table that can identify individual rows in the table separately. Out of those attributes, only one attribute is chosen as the PRIMARY KEY. The rest of the attributes become ALTERNATE KEYS.
6. **COMPOSITE KEY :-** COMPOSITE KEY is a merger of multiple columns that help in identifying each row distinctly. This distinctness is guaranteed only when the columns are combined. When the columns are taken individually, it does not promise distinctiveness. A PRIMARY KEY which is made of multiple attributes, is defined as a COMPOSITE KEY.
7. **UNIQUE KEY :-** UNIQUE KEY can also identify each row in a table uniquely like a PRIMARY KEY. But, unlike a PRIMARY KEY, a UNIQUE KEY can have only a single null value. There can be several UNIQUE KEYS in a table.

Generalization :-

- Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.
- In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.
- Generalization is more like a subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.
- In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass.

Example:-

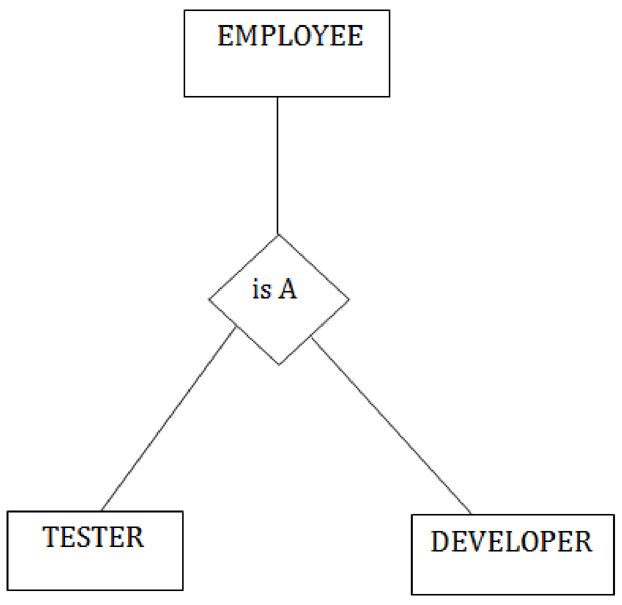


Link
[In Detail Generalization & Specialization and the difference between them]

Specialization:-

- Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.
- Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.
- Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship sets are then added.

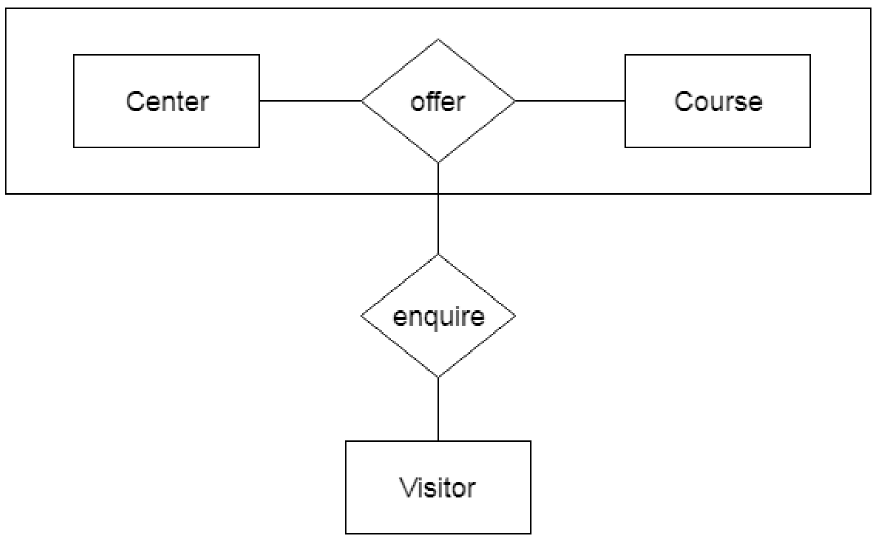
Example:-



Aggregation:-

In aggregation, the relation between two entities is treated as a single entity. In aggregation, the relationship with its corresponding entities is aggregated into a higher level entity.

For example: Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.



Data Types:-

INT	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum string length in characters - can be from 0 to 65535
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
BIT(size)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
Decimal (size, d)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

BOOL	Zero is considered as false, nonzero values are considered as true.

Integer :- INT, TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT

Between Operator:-

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.
- To display the products outside the range of the previous example, use NOT BETWEEN.

Example:-

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* BETWEEN *value1* AND *value2;*

LIKE OPERATOR:-

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator.

- **Like operator with description**

- 'a%' - starts with a
- '%a' - ends with a
- '_n%' - the second letter of name is n
- 'a_%_%' - starts with a and at least 3 character in length
- 'a%i' - starts with a and ends with i
- % - wildcard character any number
- _ - only one character which I don't know

Syntax:-

SELECT *column1, column2, ...*
FROM *table_name*
WHERE *columnN* LIKE *pattern*;

CASE STUDY

# Create table student2 using different data types
create table student2 (
s_id int,
s_name varchar(100),
gender char(1) character set ascii,
dob date,
login datetime,
register timestamp
);

# Insert values into student2 table

insert into student2 values (1, 'cef', 'm', '1999-04-14', '2022-04-22 11:34:53.44', '2020-03-15 07:31:42.23');
insert into student2 values (2, 'gef', 'f', '1999-04-14', now(), now()); # now() is use to get present date and time
insert into student2 values (3, 'efg', 'f', '1994-06-11', now(), now());
insert into student2 values (4, 'hoj', 'm', '1993-06-11', now(), now());
insert into student2 values (5, 'cee', 'm', '1999-04-14', '2022-04-22 11:34:53.44', '2020-03-15 07:31:42');
insert into student2 values (6, 'fff', 'f', '1999-04-14', now(), now());

# Update query

```
select * from student2;
update student2 set register= '2020-03-15 07:31:42'  where s_id=1;

# Between Operator

# between query using cast () keyword

select * from student2 where dob between cast('1990-02-20' as date) and cast('1995-03-15' as date);


 # between query using timestamp datatype

select * from student2 where register between '2020-02-25 00:00:00' and '2022-01-15 00:00:00';

# Current date using interval

SELECT CURRENT_DATE - interval 1 DAY;     # -1 interval
SELECT CURRENT_DATE + interval 1 DAY;     # +1 interval

# Create table student3 using decimal datatype

create table student3 (
s_id int,
s_name varchar(100),
fees decimal(8,2)
);

# Insert query
                        insert into student3 values (1, 'ann', 14522.23);
insert into student3 values (2, 'bnn', 14522.234);
                        insert into student3 values (2, 'bnn', 94862.24);
                        insert into student3 values (3, 'Mira', 15000.24),
(4, 'Aneri', 15000.24),
(5, 'Joel', 15000.24),
(6, 'Joe', 15000.24),
(7, 'Zoe', 15000.24);

# Delete a single column using condition

delete from student3 where s_name= 'bnn';



/* User Commands
To create user
grant privileges
select user from  mysql user
show grants
*/

# To create a user in database

create user aneritest@localhost identified by 'aneri';

# Grant command is used to access the privileges to the created host

grant all privileges on *.* to aneritest@localhost;

# Select user from all the mysql user

select user from mysql.user;
# Show  GRANTS command
show grants for aneritest@localhost;

# Select query from database
```

```
select user,host,account_locked,password_Expired from mysql.user;
                    select user,host,db,command from information_schema.processlist;

# command is not running needs to research on it
/* update user SET authentication_string = password(aneripatel) where user='aneritest'and host='localhost';
UPDATE user SET authentication_string='Aneri' WHERE user='aneritest@localhost';
UPDATE user SET password=password('Aneri1') WHERE user='aneritest'; */


# Creating table from the existing table using create command

select * from student4;
create table student4 select s_id, s_name from student3;


# BETWEEN  and NOT BETWEEN command

Select * from student2
where dob between '1999-04-14' and '1994-06-11';

Select * from student3
where s_name between 'Aneri' and 'Joe';

Select * from student3
where s_name not between 'Aneri' and 'Joe';

/* Like command
'a%' - starts with a,
'%a' - ends with a,
'_n%' - the second letter of name is n,
'a_%_%' - starts with a and at least 3 character in length,
'a%i' - starts with a and ends with i
 % - wildcard character any number
_ - only one character which I don't know
*/

Select * from student3 where s_name like 'a%';
Select * from student3 where s_name like '%a';
  Select * from student3 where s_name like '_n%';
```

21/02/2023

SQL ENGINE:-

The SQL Storage Engine is a software module used to create, read, and update data between the disk and memory while still maintaining data integrity (rollback journals and write-ahead logs).

Types of SQL Storage Engine

- InnoDB
- MyISAM
- MEMORY
- MRG_MYISAM
- CSV
- FEDERATED
- PERFORMANCE_SCHEMA
- ndbinfo
- BLACKHOLE
- ARCHIVE
- ndbcluster

Syntax:-

SHOW ENGINE;

-> To see which table is using which engine we can use the command

show table status;

-> When there are more than one engine or tables connect to the engine we can use the command

repair table table_name quick extended;

ALTER:-

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE- ADD COLUMN:- To add a column in a table.
Syntax:-
ALTER TABLE *table_name*
ADD *column_name datatype*;

ALTER TABLE- DROP COLUMN :- To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column).
Syntax:-
ALTER TABLE *table_name*
DROP COLUMN *column_name*;

ALTER TABLE- RENAME COLUMN :- To rename a column in a table.
Syntax:-
ALTER TABLE *table_name*
RENAME COLUMN *old_name* to *new_name*;

ALTER TABLE- ALTER/MODIFY DATATYPE:- To change the data type of a column in a table.
Syntax:- ALTER
ALTER TABLE *table_name*
ALTER COLUMN *column_name datatype*;

Syntax:- MODIFY
ALTER TABLE *table_name*
MODIFY COLUMN *column_name datatype*;

NORMALIZATION

- Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization rules divide larger tables into smaller tables and link them using relationships. The purpose of Normalisation in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

Database Normal Forms

1. 1NF (First Normal Form)
   - Each table cell should contain a single value.
   - Each record needs to be unique.

2. 2NF (Second Normal Form)
   - Rule 1- Be in 1NF.
   - Rule 2- Single Column Primary Key that does not functionally dependant on any subset of candidate key relation.

3. 3NF (Third Normal Form)
   - Rule 1- Be in 2NF.
   - Rule 2- Has no transitive functional dependencies.

4. BCNF (Boyce-Codd Normal Form)
   - BCNF is the advanced version of 3NF. It is stricter than 3NF.
   - A table is in BCNF if every functional dependency X ⟶ Y, X is the super key of the table.
   - For BCNF, the table should be in 3NF, and for every FD, LHS is a super key.

5. 4NF (Fourth Normal Form)
   - A relation will be in 4NF if it is in Boyce Codd Normal Form and has no multivalued dependency.
   - For a dependency A⟶ B, if for a single value of A, multiple values of B exists, then the relation will be a multivalued dependency.

CASE STUDY

```
/* Creat table name vehicle and inserted values in it */
create table vehicle(
vehicle_no varchar(20) primary key,
model_name varchar(45),
price decimal(10,2),
sell_price decimal(10,2));


insert into vehicle(vehicle_no,model_name,price,sell_price)
values('ssajd314231','mercedes',276473,18797),
('ssajd3142328438','mercedes1',47647,48797),
('ssajd31423341','mercedes2',67647,98797);

select  * from vehicle;

# the command  is used to see all the types of engine

show engines;

use mysql;

select table_name,engine from information_schema.tables
where table_name='vehicle';

# to select database which we want to use run this command

use aneri;

# command to change the the type of 'ENGINE' for better performance

alter table vehicle ENGINE='MyISAM';


# we can reverse it back to whichever 'ENGINE' it was before
alter table vehicle ENGINE='InnoDB';
```

```sql
# To repair the table use this command

repair table vehicle;

# when there are n number of table we can use quick extended key word for the quicker repair

repair table vehicle quick extended;

# To check the status of the table

show table status;


 # command is use to add column after particular column

 alter table vehicle
 add column description varchar(200) not null after vehicle_no;

# To check the table description use this command

 describe vehicle;

 # command is use to add column at starting by using “FIRST” keyword

 alter table vehicle
 add column vehicle_lr_no varchar(200) not null first;

# command to set a default description

 alter table vehicle alter description set default 'Car is superb';

 # to see full column description
 # collation :- Collation means assigning some order to the characters in an Alphabet, say, ASCII or Unicode etc.

 show full columns from vehicle like 'v%o';

 show full columns from vehicle;

### rename, change and modify column name and datatype
# Syntax
/*ALTER TABLE table_name
RENAME COLUMN old_name to new_name; */

alter table vehicle
rename column price to actual_price;

alter table vehicle
modify actual_price decimal(9,2);

# drop command is used to drop the particular column in combination of alter command

alter table vehicle
drop column description;

### Create view

create view merci as select vehicle_no, actual_price from vehicle where model_name='mercedes';

select * from merci;

# to lock the actual table we can only view the data cannot update

lock tables vehicle read;

# In research time unlock
```

```
unlock tables;

alter table vehicle
modify actual_price decimal(10,2);

# To check the connection_id (It is virtual name of the particular user)

select connection_id();
```

22/02/2023

Operators

1. *AND* :- The **AND** operator displays a record if all the conditions separated by **AND** are TRUE.

true	true	true
true	false	false
false	true	false
false	false	false

Syntax:-
SELECT *column1, column2, ...*
FROM *table_name*
WHERE *condition1* AND *condition2* AND *condition3 ...*;

2. ***OR*** :- The **OR** operator displays a record if any of the conditions separated by **OR** is TRUE.

true	true	true
true	false	true
false	true	true
false	false	false

Syntax:-
SELECT *column1, column2, ...*
FROM *table_name*
WHERE *condition1* OR *condition2* OR *condition3 ...*;

3. NOT :- The **NOT** operator displays a record if the condition(s) is NOT TRUE.

0	1
1	0

Syntax:-
SELECT *column1, column2, ...*
FROM *table_name*
WHERE NOT *condition*;

4. IN :- The IN operator allows you to specify multiple values in a WHERE clause.
   - The IN operator is a shorthand for multiple OR conditions.

Syntax:-
SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* IN (*value1*, *value2*, ...);

5. ANY :- returns a boolean value as a result.
   - returns TRUE if ANY of the subquery values meet the condition.

Any:-- means that the condition will be true if the operation is true for any of the values in the range.
Syntax:-
SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name operator* ANY
(SELECT *column_name*
FROM *table_name*
WHERE *condition*);

6. EXIST :- The EXISTS operator is used to test for the existence of any record in a subquery.
   - The EXISTS operator returns TRUE if the subquery returns one or more records.

   Syntax:-
   ```
   SELECT column_name(s)
   FROM table_name
   WHERE EXISTS
   (SELECT column_name FROM table_name WHERE condition);
   ```

7. ISNULL / IS NOT NULL :-
   - The IS NULL operator is used to test for empty values (NULL values).
   -The IS NOT NULL operator is used to test for non-empty values (NOT NULL values)

   Syntax:-
   SELECT CustomerName, ContactName, Address
   FROM Customers
   WHERE Address IS NULL;

   Syntax:-
   SELECT CustomerName, ContactName, Address
   FROM Customers
   WHERE Address IS NOT NULL;

8. BETWEEN :- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

   Syntax:-
   SELECT *column_name(s)*
   FROM *table_name*
   WHERE *column_name* BETWEEN *value1* AND *value2;*

Functions

1. IF :- The syntax for the IF function in a MySQL database uses the IF keyword and then takes in three parameters: the boolean expression the function is evaluating, the statement to return if the condition is true, and then the statement to return if the condition is false.

   Example:- select if(200>350, 'yes', 'no');   **Output:- no**

2. IFNULL :- IFNULL() is a conditional statement that asks whether or not the first expression is NULL. If it is, it returns the second expression and not the first.

   Example:- select ifnull(null, 5);   **Output:- 5**

3. NULLIF :- NULLIF() returns NULL when both expressions passed into the function are equal.

   Example:- select nullif("hello", "hellowww"); **Output:- hello**

Note:-
- If the values are not equal it will return first value.(hello)
- If the values are equal it will return null

CASE STUDY

#Boolean Datatype
# How to check which boolean format it has

select true, false, TRUE, FALSE;

```
# Creating student5 table

create table student5( s_id int, name varchar(250), pass boolean);

# Insert into student5 table

            insert into student5 values(1, 'aneri', true);
                                    insert into student5 values(1, 'Mira', true);
                                    insert into student5 values(3, 'mia', false);

# Query to check boolean datatype

select * from student5 where pass is true;
 select * from student5 where pass != true;

# Use of Any Operator
# Create and insert query table1 and table2

create table table1(num_value int);
                        insert into table1(num_value) values (10),(20),(25);

                                    create table table2(num_val int);
                  insert into table2(num_val) values (20),(7),(10);

# Query using Any operator

select num_val from table2 where num_val = ANY(select num_value from table1);
select num_val from table2 where num_val < ANY(select num_value from table1);

# Exist Operator
# Create table employee1 and department and insert queries

create table employee1 (e_id int not null, name varchar(50));
                           insert into employee1 values(1, 'Mia'),
                           (2, 'John'),
                           (3, 'Victor');
                           insert into employee1 (e_id) values(4);

                          create table department (dep_id int not null, dep_name varchar(50), e_id int);
                          insert into department values(1, 'Calculation', 1),(2, 'IT', 3);

# Query using exist operator

select name from employee1 where exists (select * from department where employee1.e_id= department.e_id );

/* IN and NOT IN operator use */
                              select * from employee1
where e_id not in (select e_id from department);


# IS NULL / IS NOT NULL Operator

select * from employee1 where name is NOT NULL;
select * from employee1 where name is  NULL;

/* STRING COMPARISON STRCMP()
If string1 = string2, this function returns 0
If string1 < string2, this function returns -1
If string1 > string2, this function returns 1
 */
select strcmp('aneri', 'dkdkjgk');
select strcmp('aneri', 'AAAAA');

# BETWEEN Operator using date
# create table student2
create table student2 (
                                                s_id int,
s_name varchar(100),
                                                gender char(1) character set ascii,
                                                dob date,
                                                login datetime,
                                                register timestamp );
```

# insert values into student2 table

```
insert into student2 values (1, 'cef', 'm', '1999-04-14', '2022-04-22 11:34:53.44', '2020-03-15 07:31:42.23');
insert into student2 values (2, 'gef', 'f', '1999-04-14', now(), now() );
insert into student2 values (3, 'efg', 'f', '1994-06-11', now(), now() );
insert into student2 values (4, 'hoj', 'm', '1993-06-11', now(), now() );
insert into student2 values (5, 'cee', 'm', '1999-04-14', '2022-04-22 11:34:53.44', '2020-03-15 07:31:42');

update student2 set register= '2020-03-15 07:31:42'  where s_id=1;
```

# Query using Between Operator with date data type we use **CAST**

```
select * from student2 where register between cast('2023-02-20' as date) and cast('2020-03-15' as date);
```

# Query for data type timestamp

```
select * from student2 where register between '2022-02-25 00:00:00' and '2019-01-15 00:00:00';
```

if

```
select if(200>350, 'yes', 'no');
select if(200=200, 'yes', 'no');
```

# ifnull

```
select ifnull(null, 5);
select ifnull(null, "sql");
```

# nullif

```
select nullif("hello", "helloowww");
select nullif("hello", "hello");
```

23/02/2023

CASE:- The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

- If there is no ELSE part and no conditions are true, it returns NULL.

Syntax:-

```
CASE
   WHEN condition1 THEN result1
   WHEN condition2 THEN result2
   WHEN conditionN THEN resultN
   ELSE result
END;
```

JOIN :- JOIN means *to combine something*. In the case of SQL, JOIN means "to combine two or more tables".

- The SQL JOIN clause takes records from two or more tables in a database and combines it together.

ANSI standard SQL defines five types of JOIN :

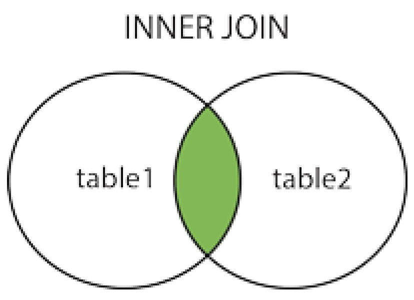
1. Inner Join:- The INNER JOIN keyword selects records that have matching values in both tables.

   Syntax:-

   ```
   SELECT column_name(s)
   FROM table1
   INNER JOIN table2
   ON table1.column_name = table2.column_name;
   ```

Multi-Table Join Syntax:-

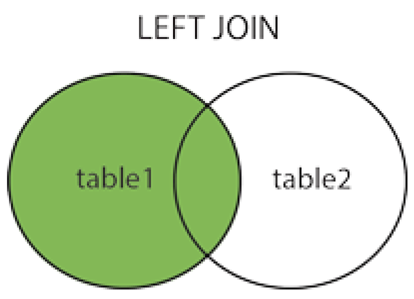
```
SELECT column-names
 FROM table-name1
INNER JOIN table-name2 ON column-name1 = column-name2
INNER JOIN table-name3 ON column-name3 = column-name4
INNER JOIN table-name4 ON column-name5 = column-name6
...
WHERE condition
```



2. Left Join:- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

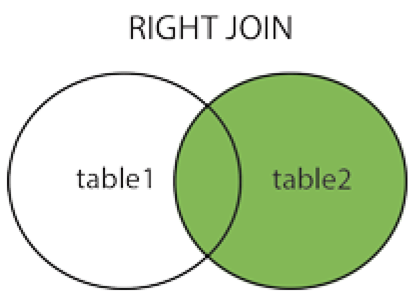
   Syntax:-

   ```
   SELECT column_name(s)
   FROM table1
   LEFT JOIN table2
   ON table1.column_name = table2.column_name;
   ```



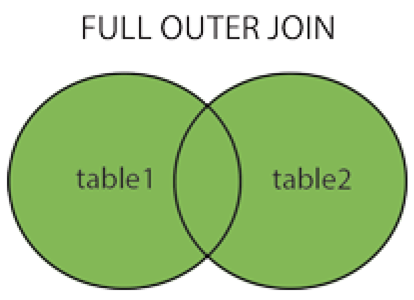
3. Right Join:- The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

   Syntax:-
   SELECT *column_name(s)*
   FROM *table1*
   LEFT JOIN *table2*
   ON *table1.column_name* = *table2.column_name*;

4. Full Outer Join:- The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.
   - FULL OUTER JOIN and FULL JOIN are the same.

   Syntax:-
   SELECT *column_name(s)*
   FROM *table1*
   FULL OUTER JOIN *table2*
   ON *table1.column_name* = *table2.column_name*
   WHERE *condition*;

5. Self Join:- A self join is a regular join, but the table is joined with itself.

   - *T1* and *T2* are different table aliases for the same table.

   Syntax:-
   SELECT *column_name(s)*
   FROM *table1 T1*, *table1 T2*
   WHERE *condition*;

6. Cross Join:- The SQL CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table if no WHERE clause is used along with CROSS JOIN.This kind of result is called as Cartesian Product.

   Syntax:
   SELECT *
   FROM table1
   CROSS JOIN table2;

   - Cross join will give multiple values linked with each other.

UNION:- The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

  Syntax:-
  SELECT *column_name(s)* FROM *table1*
  UNION
  SELECT *column_name(s)* FROM *table2*;

UNION ALL:- The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL.

Syntax:-
SELECT *column_name(s)* FROM *table1*
UNION ALL
SELECT *column_name(s)* FROM *table2*;

Distinct:- The SELECT DISTINCT statement is used to return only distinct (different) values.

Syntax:
SELECT DISTINCT *column1*, *column2, ...*
FROM *table_name*;

Order By with ASC|DESC:-

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

Syntax:
SELECT *column1*, *column2, ...*
FROM *table_name*
ORDER BY *column1, column2, ...* ASC|DESC;

Group By:- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

- The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

Syntax:
SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
ORDER BY *column_name(s);*

CASE STUDY

# query using Case

```
select case 1 when 1 then 'one 'when 2 then 'two' else 'more' end;
select case binary 'B' when 'a' then 1 when 'B' then 2 else 'enjoy' end;
```

# Create, Insert table and Case query

```
create table result(s_id int not null, name varchar(100), score int);
insert into result values(1, 'aneri', '85'),
(2, 'mark', 39),
(3, 'john', 99),
(4, 'jenny', 79),
(5, 'liza', 55),
(6, 'anna', 56),
(7, 'callum', 69),
(8, 'ivan', 90);
# Case query
SELECT *,
 CASE
  WHEN score >= 94 THEN "A"
  WHEN score >= 85 THEN "A-"
  WHEN score >= 75 THEN "B+"
  WHEN score >= 70 THEN "B"
  WHEN score >= 65 THEN "B-"
  WHEN score >= 60 THEN "C+"
  WHEN score >= 55 THEN "C"
  WHEN score >= 50 THEN "C-"
```

```
    WHEN score >= 45 THEN "D+"
    WHEN score >= 35 THEN "D"
    ELSE "F"
  END AS grade
FROM result;


# Order By with desc
SELECT name,
  CASE
   WHEN score >= 94 THEN "A"
    WHEN score >= 85 THEN "A-"
    WHEN score >= 75 THEN "B+"
    WHEN score >= 70 THEN "B"
    WHEN score >= 65 THEN "B-"
    WHEN score >= 60 THEN "C+"
    WHEN score >= 55 THEN "C"
    WHEN score >= 50 THEN "C-"
    WHEN score >= 45 THEN "D+"
    WHEN score >= 35 THEN "D"
    ELSE "F"
  END AS grade
FROM result
ORDER BY score DESC;


/* Example by mates
select case  when 2<1 then 1 when 3>2 then 2 else 'enjoy' end;

 SELECT CASE, CASE X is evaluated
 SELECT
 CASE 3
  WHEN 1 THEN 'one'
  WHEN 2 THEN 'two'
  WHEN 3 THEN 'three'
  ELSE 'more' END;

create table salary(id int primary key, name varchar(255), sex enum("m","f"),salary int);
insert into salary values(1,"A","m",2500),(2,"B","f",1500),(3,"C","m",5500),(4,"D","f",500);
select * from salary;
###SWAP m with F
#1 possibility
UPDATE salary set sex = if(sex="f","m","f");

#2 possibility
UPDATE salary
    SET sex  = (CASE WHEN sex = 'm'
        THEN  'f'
        ELSE 'm'
        END);
*/

/* Joins Query by Jinesh
select distinct o.order_id as OrderNumber, c.customer_name as CustomerName
from order_new o
left outer join customer_new c
on c.customer_id = o.customer_id

select o.order_id as OrderNumber, c.customer_name as CustomerName
from order_new o
left join customer_new c
using (customer_id);

select o.order_id as OrderNumber, c.customer_name as CustomerName
from order_new o
left outer join customer_new c
```

```sql
on c.customer_id = o.customer_id
union
select o.order_id as OrderNumber, c.customer_name as CustomerName
from order_new o
right join customer_new c
using (customer_id);

select  o.order_id as OrderNumber, c.customer_name as CustomerName
from order_new o inner join customer_new c
on c.customer_id = o.customer_id;

select  o.order_id as OrderNumber, c.customer_name as CustomerName
from order_new o cross join customer_new c
on c.customer_id = o.customer_id;
*/

# Create employee1 and department table
create table employee1 (e_id int not null, name varchar(50));
                        insert into employee1 values(1, 'Mia'),
                        (2, 'John'),
                        (3, 'Victor');
                        insert into employee1 (e_id) values(4);

                      create table department (dep_id int not null, dep_name varchar(50), e_id int);
                       insert into department values(1, 'Calculation', 1),(2, 'IT', 3);

select * from employee1;
select * from department;

# Inner Join

select d.dep_name as DepName, e.name as EmpName
from department d
inner join employee1 e
on e.e_id = d.e_id;

# Inner Join with using keyword

select d.dep_name as DepName, e.name as EmpName
from department d
inner join employee1 e
using (e_id);

# Use of Distinct keyword to retrieve distinct values

select distinct d.dep_name as DepName, e.name as EmpName
from department d
inner join employee1 e
on e.e_id = d.e_id;

# Left join or left outer join

select d.dep_name as DepName, e.name as EmpName
from department d
left outer join employee1 e
using (e_id);

# Right join or Right outer join

select d.dep_name as DepName, e.name as EmpName
from department d
right join employee1 e
using (e_id);

# Cross join
```

```
select d.dep_name as DepName, e.name as EmpName
from department d
cross join employee1 e;




# Full outer join IT IS NOT SUPPORTED IN SQL

select department.dep_name as DepName,  employee1.name as EmpName
from department
FULL OUTER  JOIN employee1
on  employee1.e_id = department.e_id;

# multi join query

create table student16(id int, name varchar(50),branch varchar(50));
create table marks(id int, marks int);
create table attendance(id int, attendance int);

insert into student16 values(1,'anu','cse');
insert into student16 values(2,'mia','ece');
insert into student16 values(3,'sam','ece');
insert into student16 values(4,'kae','cse');

insert into marks values(1,95);
insert into marks values(2,85);
insert into marks values(3,80);
insert into marks values(4,65);

insert into attendance values(1,75);
insert into attendance values(2,65);
insert into attendance values(3,80);
insert into attendance values(4,80);



select s.id, name, marks, attendance
from student16 as s
inner join
marks as m
on s.id=m.id
inner join
attendance as a
on m.id=a.id
where a.attendance>=75;


# delete the record example
/*delete o,c
from order_new o
inner join customer_new c
on o.customer_id=c.customer_id
where o.order_id is NULL;
*/


# UNION and UNION ALL Condition

create table student14(s_id int not null, s_name varchar(100), age int);
create table teacher(t_id int not null, t_name varchar(100), age int);

insert into student14 values(1, 'anna', 16),
(2, 'bnna', 17),
(3, 'cnna', 15),
(4, 'dnna', 19);
```

```
insert into teacher values(1, 'avca', 35),
(2, 'hjma', 45),
(3, 'zyna', 26),
(4, 'dina', 36);

#UNION
select age
from teacher
union
select age
from student14;

# UNION ALL

select age
from teacher
union all
select age
from student14;

# Ascending descending example by jinesh
select * from customer_new order by customer_id asc, customer_name desc;

/* Query by Jinesh
            create table employee(eid int, emp_name varchar(250),  income int)
create table employee1(eid int, emp_name varchar(250),  income int, city varchar(236))
insert into employee1 values(1,'asjhdskj',7163,'a'),
(2,'asjhdskj',2734163,'a'),
(3,'asjhdskj',73,'b'),
(4,'asjhdskj',7161233,'a'),
(10,'asjhdskj',716332,'b')

select city,min(income) as "Minimum income"
from employee1
group by city
having min(income)>73 */

# Create and Insert query for product table

create table product (p_id int not null,
 p_name varchar(200),
sup_id int,
p_unit int,
price decimal(10,2));

insert into product values(1, 'book', 34, 80, 65789.95),
(2, 'sketch', 42, 69, 896789.95),
(3, 'stationary', 38, 90, 965789.95),
(4, 'schoolbag', 64, 100, 321235.55),
(5, 'lunchbox', 98, 200, 569389.95);

# Use of min()

select min(price) as SmallestPrice
from product;

# Use of max()

select p_name, max(p_unit) as No_of_Products
from product;

# use of HAVING

select p_name, sum(price) as 'TotalAmount' from product group by p_name having sum(price)>65789.95;
select p_name, avg(price) as 'TotalAmount' from product group by p_name having sum(price)>65789.95;



/* create table emp_business (e_id int, e_name varchar(250), occupation varchar(250),working_date date, working_hours int)

insert into emp_business values(1,'aksjdhgkjsad','IT_DB',null, 5),
(2,'aksdjdhgkjsad','IT_Com',null, 7),
(3,'aksjdhgkjsad','IT_tech',null, 3),
(4,'aksjdhgkjsad','IT_HR',null, 7),
(5,'aksjdhgkjsad','IT_innovat',null, 6),
(6,'aksjdhgkjsad','IT_resarch',null, 4)

select occupation ,sum(working_hours) as "total working hours" from emp_business group by occupation having sum(working_hours)>6;
select occupation ,avg(working_hours) as "total working hours" from emp_business group by occupation having avg(working_hours)>6;
*/
```

24/02/2023

ALL- It is always used with the combination of different keywords.

Regular Expression:-

Operator	**Operator Name**	**Function**
(.)	Any character – Dot Quantifier	Matches any single character in the character set of the database.
(*)	Asterisk or Star Quantifier	Matches zero or more occurrences of the subexpression/ strings preceding to it.
(+)	Plus Quantifier	Matches one or more occurrences of the subexpression/ strings preceding to it.
(?)	Question mark Quantifier	Matches zero or one occurrence of the subexpression/ strings preceding to it.
[ABC] / [abc]	Matching Character List	Matches any character mentioned in the list.
[^ABC] / [^abc]	Non-Matching Character List	Matches any character except the ones mentioned in the list.
[0-9]	Digit List	Matches any digit from 0 to 9.
{a}	Exact count Interval	Matches exact ‘a’ occurrences of subexpression or string preceding to it.
{a,}	At Least one count Interval	Matches at least ‘a’ occurrences of subexpression or string preceding to it.
{a,b}	Between count Interval	Matches at least ‘a’ occurrences of subexpression or string preceding to it but not more than ‘b’ occurrences.
(^)	Caret Quantifier	Matches an expression only if it occurs at the beginning of a line.
($)	Dollar or End Quantifier	Matches an expression only if it occurs at the end of a line.
(|)	Vertical Bar Quantifier	It is used to isolate different alternatives in a regular expression.
[[:class:]]	Class Quantifier	Matches the character class , for example , matches [[:digit:]] to digits, [[:space:]] to space, [[:alnum:]_] to alpha numerics, etc.

Rlike:- This operator in MySQL is used to perform a pattern match of a string expression against a pattern.

INDEX:-
- The CREATE INDEX statement is used to create indexes in tables.
- Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Syntax:-
CREATE INDEX *index_name*
ON *table_name* (*column1*, *column2*, ...);
DROP INDEX:- The drop index statement is used to delete an index in a table.

Syntax:-
DROP INDEX *index_name* ON *table_name*;

To see the index
- show index from table_name where column_name=’name’;

AUTO INCREMENT:- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

Syntax:-
```
CREATE TABLE Persons (
   Personid int NOT NULL AUTO_INCREMENT,
          LastName varchar(255) NOT NULL,
          FirstName varchar(255),
          Age int,
   PRIMARY KEY (Personid)
);
```

PARTITION:- Partition can only be applied while creating a table.

Syntax:-
```
SELECT column_name,
    window_function (expression) OVER (PARTITION BY column name)
FROM table;
```

CASE STUDY

# Query for REGULAR EXPRESSION using keyword REGEXP

```
select * from employee1;

select * from employee1 where name regexp '^[MV]';

select * from employee1 where name regexp '[an]$';
select * from employee1 where name regexp 'a$';

select * from employee1 where name regexp '^.{3}$';

select * from employee1 where name regexp 'c';

select * from employee1 where name regexp '^[^mv]' and '^[^m^v]';
```

# Match and CHECK

```
select ('B' REGEXP '[A-Z]' ) AND  ('M' REGEXP '[D-Z]' )  AS MATCH_;

select REGEXP_LIKE('a', '[A-Z]', 'i') as CHECK_;
```

#REGEXP using substring

```
SELECT REGEXP_SUBSTR('Cat Cut Cot', 'C.t', 2) Result;
SELECT REGEXP_SUBSTR('Cat Cut Cot', 'C.t', 1) Result;
SELECT REGEXP_SUBSTR('Cat Cut Cot', 'C.t', 1, 3) Result;
```

#REGEXP_REPLACE(expr, pat, repl[, pos[, occurrence[, match_type]]])

```
SELECT REGEXP_REPLACE('abc def ghi jk mn abc def', 'def', 'X', 2, 2);
```

# Select the price which have 1,2,3 in the figure

```
select * from product where price regexp '[1-3]';
select * from product where price regexp '^[8]|^[9]';
```

# INDEX

```
select * from employee1;

create index name_index on employee1(name);

show index from employee1 where column_name = 'name';

select * from employee1 where name='Mia';


/* code by Jinesh for auto increment
 create table example_temp(id int auto_increment primary key,
name varchar(250) not null,
age int not null)

insert into example_temp(name, age) values('John',25)
insert into example_temp(name, age) values('Jane',30);
*/

# To get the last value inserted

select last_insert_id()

/* Code by Jinesh on  Partition
create table sales(id int not null auto_increment,
sale_date date not null,
amount DECIMAL(10,2) not null,
PRIMARY KEY(id,sale_date))
partition by range(year(sale_date))
(
partition p0 values less than (2010),
partition p1 values less than (2011),
partition p2 values less than (2012),
partition p3 values less than (2013),
partition p4 values less than (2014),
partition p5 values less than (2015),
partition p6 values less than (2016),
partition p7 values less than (2017),
partition p8 values less than (2018),
partition p9 values less than (2019),
partition p10 values less than MAXVALUE
)
*/

# Copy from one table to another table

insert into sales (id,sale_date,amount)
select id, sale_date,amount from sales_party;

/* Code by Jinesh using Partition with over()
select *, row_number() over (partition by sale_date) as row_num from sales

create table person (name varchar(45) not null,
product varchar(45),
country varchar(45),
year int)

insert into person(Name,product,country,year)
values('as','asdasd','asdasd',2015),
('as','asdasd','asdasd',2016),
('as','asdasd','asdasd',2017),
('as','asdasd','asdasd',2015),
('as','asdasd','asdasd',2018)
select *, row_number() over (partition by year) as row_num from person
*/
```

```
/* Example by mate on Partiotion with over()

CREATE TABLE sales2(
id INT NOT NULL AUTO_INCREMENT,
sale_date DATE NOT NULL,
amount DECIMAL(10,2) NOT NULL,
PRIMARY KEY(id,sale_date));
INSERT INTO sales2 (id, sale_date, amount) VALUES
(1, '2014-11-03', 54353),
(2, '2017-01-04', 2576727),
(3, '2016-08-08', 757654),
(4, '2014-04-06', 324235),
(5, '2023-05-03', 23334),
(6, '2019-12-01', 2342134),
(7, '2020-05-03', 24351534),
(7, '2020-07-12', 2153574),
(7, '2020-09-17', 1121534);
SELECT * from sales2;

SELECT * , ROW_NUMBER() OVER (PARTITION BY YEAR(sale_date)) AS row_num FROM sales2;
*/


/* Partition example with over() keyword

 create table student_score(name varchar(50),score int);

insert into student_score
values ('Alice',90),
('Bob',85),
('carl',75),
('Dave',80),
('eric',90);

select Name, score, rank() OVER (ORDER BY SCORE DESC) AS SCHOOL_RANK from student_score;

select Name, score, DENSE_rank() OVER (ORDER BY SCORE DESC) AS SCHOOL_RANK from student_score;

select name, score, SCHOOL_RANK, (select name, score, DENSE_rank() OVER (ORDER BY SCORE DESC) AS SCHOOL_RANK from  student_score) from
student_score; # not working

SELECT NAME , SCORE , SCHOOL_RANK FROM (
select Name, score, rank() OVER (ORDER BY SCORE DESC) AS SCHOOL_RANK from student_score ) AS RANKED_SUDENTS WHERE SCHOOL_RANK=3
 */
```