Introduction to Database and SQL :-

A database is an organized collection of data so that it can be Easily accessed. To manage these databases, Database Management Systems (DBMS) are used.

Type of DBMS :-

In general, there are two common types of databases -

- Non - Relational
- Relational

Non- Relational Database Manggement System (Non-RDBMS) :-

In Non- RDBMS, data is stored in key - value Pairs.

Example -

```
{
    "id" : 1,
    "name" : "John",
    "age" : 25
}

{
    "id" : 2,
    "name" : "Marry",
    "age" : 22
}
```

Here, Customers data are stored in key - value Pairs.

Commonly used Non - RDBMS : MongoDB, Amazon DynamoDB, Redis Etc.

Relational Database Management System (RDBMS) :-

In RDBMS, data is stored in tabular format.
Example -

Customer_id	first_name	last_name	Phone	Country
1	John	Doe	817 - 646 - 6248	USA
2	Robert	Luna	217 - 464 - 4824	USA
3	David	Robinson	912 - 726 - 8666	UK
4	John	Reinhardt	812 - 762 - 8211	UK
5	Betty	Taylor	871 - 721 - 8122	UAE

Here, Customer is a table inside the database.

The first row is the Attributes of the table. Each row After that Contains the data of a Customer.

In RDBMS, two or more tables may be Related to Each other. Hence the term "Relational" means.

Introduction to SQL :-

Structured Query Language (SQL) is a Standard Query Language that is used to work with Relational databases.

We use SQL to :

- Create databases
- Create tables in a databases
- Read data from a table
- insert data in a table
- Update data in a table
- delete data from a table
- delete databases tables
- delete databases

and many more databases operations.

SQL Example of Read data from a table :-

Let's take a look at an Example- ,

```
SELECT first_name , last_name FROM Customers ;
```

10/10/22

SQL SELECT :-

The SQL SELECT Statement is Used to Select (Retrieve) data from a database table.

Example -

```
SELECT first_name , last_name
    FROM customers ;
```

customer_id	first_name	last_name	age	Country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

(figure 1.) After run

first_name	last_name
John	Doe
Robert	Luna
David	Robinson
John	Reinhardt
Betty	Doe

SQL SELECT ALL :-

To Select all Columns from a database table, we use the * character.

Example -

```
SELECT *
FROM Customers ;
```

Here, the SQL Command Select all Columns of the Customers table.

After run "table by figure 1".

Customer_id	first_name	last_name	age	Country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

SQL SELECT WHERE Clause :-

A SELECT Statement can have an optional WHERE clause. The WHERE clause allows us to fetch Records from a database table that matches Specified Condition.

Example -

```
SELECT *
FROM Customers
WHERE last_name = 'Doe' ;
```

Here, the SQL Command Selects all Customers from the Customers table with last_name Doe.

After run "table by figure 1" -

Customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
5	Betty	Doe	28	UAE

SQL Operators :-

The WHERE Clause uses operators to construct conditions. Some of the commonly used operators are :

1. Equal to Operator (=) :-

```
SELECT *
FROM Customers
WHERE first_name = 'John';
```

This SQL Command Selects all Customers from the Customers table having first_name John.
After run "table by Figure 1" -

Customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
4	John	Reinhardt	25	UK

2. Greater than (>) :-

```
SELECT *
FROM Customers
WHERE age > 25;
```

This SQL Command Selects all Customers from the Customers table having age greater than 25.

After run " table by figure 1 " -

Customer_id	first_name	last_name	age	Country
1	John	Doe	31	USA
5	Betty	Doe	28	UAE

SQL AND, OR and NOT Operators :-

The AND, OR and NOT Operators in SQL are used with the WHERE And HVING clauses.

SQL AND Operator :-

The SQL AND Operator Selects data if all conditions are True.

Example -

```
SELECT first_name, last_name
FROM Customers
WHERE Country = 'USA' AND Last_name = "Doe";
```

After run " table by figure 1 "-

first_name	last_name
John	Doe

Here, the SQL command Selects first_name and last_name of All Customers where the Country is USA and last_name as Doe from the Customer table.

SQL OR Operator :-

The SQL OR operator Select data if any one Condition is true.

Example-

```
SELECT first_name, last_name
FROm Customers
WHERE Country = 'USA' OR last_name = 'Doe';
```

After run " table by figure 1 "-

first_name	last_name
John	Doe
Robert	Luna
Betty	Doe

SQL NOT Operator :-

The SQL NOT operator selects data if the given condition is FALSE.

Example-

```
SELECT first_name, last_name
FROM customers
WHERE NOT Country = 'USA';
```

After run " table by figure 1 Page No. 2".

first_name	last_name
David	Robinson
John	Reinhardt
Betty	Doe

Here the SQL command selects first_name and last_name of all customers where the country is not USA from the customer tables.

Combining Multiple operators :-

Let's Suppose we want to select customers where the country is Either USA or UK, and the age is less than 26.

Example-

```
SELECT *
FROM customers
WHERE (Country = 'USA' OR Country = 'UK') AND age<26;
```

After run " table by figure 1 Page No. 2" -

Customer_id	first_name	last_name	age	country
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

SQL SELECT DISTINCT Statement :-

The SQL SELECT DISTINCT statement selects unique rows from a database table.

Example -

```
SELECT DISTINCT Country
FROM Customers ;
```

After run " table by figure 1 Page No. 2" -

Country
USA
UK
UAE

Here, the SQL Command selects unique countries from the Customers table.

DISTINCT with COUNT ;-

If we need to count the numer of unique rows, we can use the count() function with DISTINCT.

```
SELECT COUNT (DISTINCT Country)
FROM Customers ;
```

After run "table by figure 1 Page no. 2"-

```
Count (DISTINCT Country)
3
```

Here, the SQL command returns the count of unique Countries.

SQL IN Operator ;-

The IN operator is used with the WHERE clause to match values in a list.

Example-

```
SELECT first_name , Country
FROM Customers
WHERE Country IN ('USA', 'UK');
```

Here, the SQL command selects rows if the Country is either USA OR UK.

After run " table by figure 1 Page No. 2 "-

first_name	country
John	USA
Robert	USA
David	UK
John	UK

SQL BETWEEN Operator :-

The Between operator is used with the where clause to match values in a range.

Example-

```
SELECT item, amout
FROM Orders
WHERE amount BETWEEN 300 AND 500;
```

ORDERS →

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

Figure 2

After run -

item	amount
Keyboard	400
Mouse	300
Keyboard	400

SQL IS NULL :-

The IS NULL Condition is used to select rows if the specified field is NULL.

Example-

```
SELECT *
FROM Employee
WHERE Email IS NULL;
```

Here, the SQL command selects Employees who do not have Email.

Table : Employee :- (figure - 3)

Employee_id	first_name	last_name	department	Email
1	Peter	Doe	operations	Peter@gmail.com
2	Megan	Morel	finance	NULL
3	Rose	Bailey	Operations	rose@gmail.com
4	Linda	Bailey	finance	NULL
5	Mary	Doe	Sales	NULL

Then Run the code and see this O/P - 10/06/22

Employee_id	first_name	last_name	department	Email
2	Megan	Morel	finance	NULL
4	Linda	Baley	finance	NULL
5	Mary	Doe	Sales	NULL

IS NOT NULL :-

In SQL, Is NOT NULL condition is used to selects rows if the specified field is NOT NULL.
Example-

```
SELECT *
FROM Employee
WHERE Email IS NOT NULL;
```

Here, the SQL command selects Employee who have Emails.

After run " figure 3 by Page No. 7 is Employee".

Employee_id	first_name	last_name	department	Email
1	Peter	Doe	Operations	Peter@gmail.com
3	Rose	Bailey	Operations	rose@gmail.com

IS NULL with COUNT() :-

We can Use the COUNT() function with IS NULL to count the Number of rows with an Empty fields.
Example-

```
SELECT COUNT(*)
FROM Employee
Where Email IS NULL;
```

After run " figure 3 by Page No 7 is Employee" -

Count (*)
3

SQL MAX() Function :-

The MAX() function Returns the maximum value of a Column.

Example-

```
SELECT MAX (age)
FROM Customers;
```

After run " figure 1 by Page No. 2 is Customers" :

MAX (age)
31

SQL MIN () function :-

The MIN () function Returns the minimum value of a Column.

Example-

```
SELECT MIN (age)
FROM Customers;
```

After run " figure 1 by Page No. 2 is Customers" -

MIN(age)
22

MAX() and MIN() in Nested SELECT :-

As we know, the MAX() function returns the minimum value.
However, if we want to select the whole row containing that value, we can use the nested SELECT statement like this.
Example -

```
SELECT *
FROM Customers
WHERE age = (
    SELECT MIN(age)
    FROM Customers
);
```

Here, the SQL command selects all the customers with the smallest age.

After run, " figure 1 by Page No. 2 is Customers" -

Customer_id	first_name	last_name	age	Country
2	Robert	Lung	22	USA
3	David	Robinso	22	UK

SQL COUNT() :-

The COUNT() function returns the number of rows in the result set.

Example.

```
SELECT COUNT (*)
FROM Customers ;
```

After run, " figure 1 by Page No. 2 is Customers"-

COUNT (*)
5

SQL SUM() function :-

The SQL SUM() function is used to calculate the sum of numeric values in a column.

Example -

```
SELECT SUM (amount) AS total_sales
FROM Orders.
```

After run, " figure 2 by Page No. 6 is Orders "-

total_sales
13350

SQL AVG() function :-

The SQL() function is used to calculate the average of numeric values in a column.

Example-

```
SELECT AVG(age) AS average_age
FROM Customers;
```

Here, the SQL command returns the average age of all customers.

After run, " figure 1 by Page No. 2 Customers" -

average_age
25.6

Let's take a look at another example -

```
SELECT DISTINCT (customer_id, AVG(amount) AS
average_spends
FROM Orders
GROUP BY customer_id;
```

After run, " figure 2 by Page No. 6 is order" -

Customer_id	average_spends
1	400
2	250
3	12000
4	350

SQL ORDER BY Clause :-

The ORDER BY clause is used to short the result set in either ascending or descending order.

Example-

```
SELECT *
FROM Customers
ORDER BY first_name;
```

After run, " figure 1 By Page number 2 is Customers"-

Customer_id	first_name	last_name	age	Country
5	Betty	Doe	28	UAE
3	David	Robinson	22	UK
1	John	Doe	31	USA
4	John	Reinhardt	25	UK
2	Robert	Luna	22	USA

ORDER BY ASC (Ascending Order) :-

we can use the ASC keyword explicitly to sort selected records in ascending order.

Example-

```
SELECT *
FROM Customers
ORDER BY age ASC;
```

10/06/22

the SQL Command select ascending order by age.

After run, "figure 1 by Page no. 2 is Customers"-

Customer_id	first_name	last_name	age	Country
2	Robert	Luna	22	USA
3	David	Robinson	22	Uk
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE
1	John	Doe	31	USA

ORDER BY with WHERE :-

We can also use ORDER BY with the Select WHERE Clause.

Example-

```
SELECT last_name, age
FROM Customes
WHERE NOT Country = 'UK'
ORDER BY last_name DESC ;
```

Here, The SQL Command first select last_name and age fields from the Customer table if their Country is not UK.

After run, "figure 1 by Page NO. 2 is Customers"-

last_name	age
Luna	22
Doe	31
Doe	28

SQL Group BY :-

In SQL, the GROUP BY clause is used to group rows by one or more columns.

Example-

```
SELECT Country, COUNT (*) AS number
FROM Customers
GROUP BY Country;
```

After run, " figure 1 By Page No. 2 is Customers"-

Country	Number
UAE	1
USA	2
UK	2

SQL LIKE Operators :- The LIKE operator in SQL is used with the WHERE clause to get a Result set that matches the given string Pattern.

Example-

```
SELECT *
FROM Customers
WHERE Country LIKE 'UK';
```

After Run, " figure 1 by Page No. 2 is Customers"-

Customer_id	first_name	last_name	age	Country
3	David	Robinson	22	Uk
4	John	Reinhardt	25	Uk

Data model :-

Data model gives us an idea that how the final system will looke like after its complete implementation. It define the data Elements and the Relationship between the data Elements. Data model are used to show how data is stored, connected, accessed and updated in the database management system.

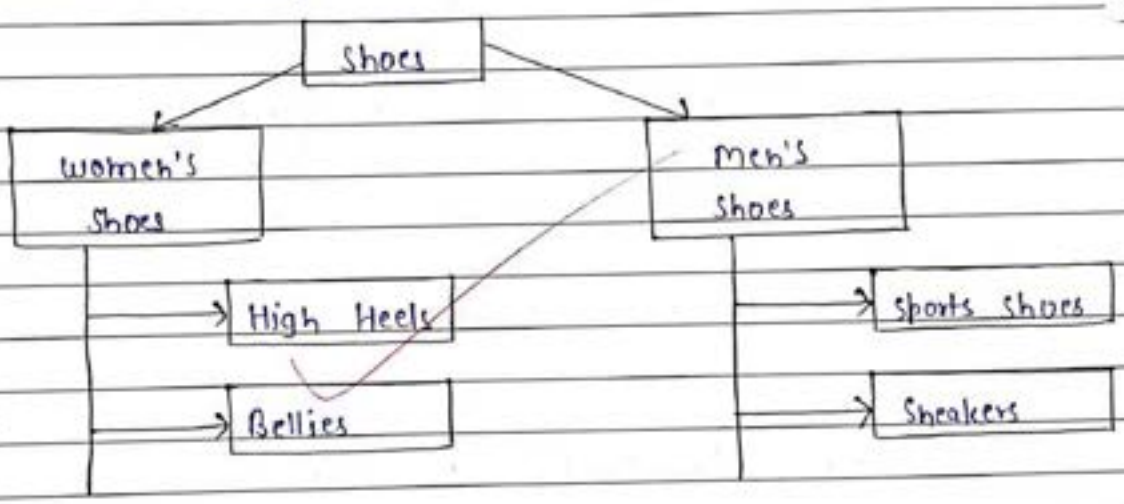
Here, we use a set of symble and text the Represent the information show that members of the organigation can communicate and understand it. Though there are many data models being used nowadays but the Relational model is the most widely used model.

Some of data models in DBMS are :

1. Hierarchical model
2. Network model
3. Entity - Relationship model
4. Relational model
5. Object - Oriented Data model
6. Object - Relational Data model
7. flat data model
8. Semi - structured Data model
9. Associative data model
10. Context data model

1. Hierarchical model :-

Hierarchical model was the first DBMS model. This model organises the data is the hierarchical tree structure.
Example - we can Represent the Relationship between the shoes present on a shoping website in the following way :



2. Network model :-

This model is an Extension of the Hierarchical model. It was the most Popular model before the Relational model.

3. Entity - Relationship model :-

This or simply ER model is a high-level data model digram. In this model, we represent the real-world problem in the Pictorial from the make it easy for the Stakeholders to Understand.

4. Relational model :-

Relational model is the most widely used model. In this model, the data is maintained in the form of a two-dimentional table. All the information is stored in the form of row and columns.

5. object - Oriented data model :-

The Real - world Problems are more closely Represented through the object - Oriented data model. In this model, both the data and Relationship are present in a Single structure known as a object.

6. object - Relational data model :-

As the Name suggests it is combination of both the Relational model and the object - oriented model. This model was built to fill the trap between object-oriented model and the Relational model.

7. flat data model :-

It is a Simple model in which the database is Represented as a table Consisting of rows and columns.
To access any data, the Computer has to Read the Entire table. This makes the models slow and inefficient.

8. Semi - structured model :-

Semi - structured model is an Evolved form of the Relational model. We can not differentiate between data and schema in this model.

9. Associate data model :-

Associate data model is a model in which the data is divided into two Parts. Everything which has independent Existence is called as an Entity and the Relationship among these Entities are called association.

The data divided into two parts are called items and links.

Item - Item Contain the name and the identifire (some Neumeric value).

Links - Links Contain the identifire, source, verb and subject.

10. Context Data model :-

Context data model is a collection of several models. This Consist of models like Networks models, Relational models etc. using this model we can do various types of tasks which are not possible using any model alone.

Concept of DBA (Database Administrator) :-

A DBA is a Person or group in charge of implementing DBMS in an organization. The DBA job requires a high degree of technical Expertise. DBA Consists of a team of people rather than just one Person.

The Primary role of DBA is as follows :

- Database design
- Performance issues
- Database accessibility
- Capacity issues
- Data Replication
- Table maintenance

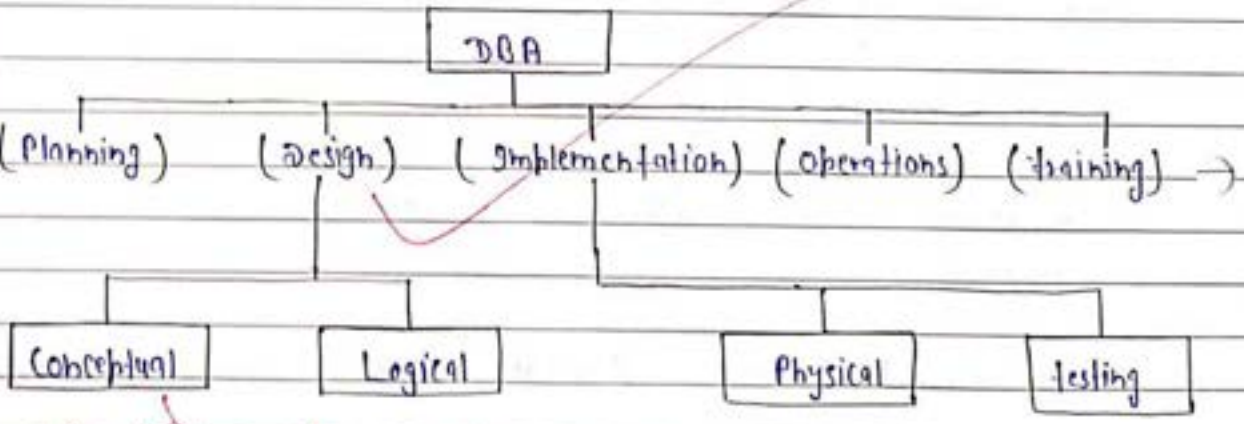
Responsibilities of DBA :-

- Makes the decision Concerning the Content of the database.
- Plans the storage Structure and access strategy.
- Provides the Support to the users.
- Define the Security and integrity checks.
- Monitoring the performance and Responding to the changes in the Requirments.

Skill Requird for DBA :-

- Database designing
- Knowledge of structured Query Language (SQL)
- Know about distributed architecture.
- Knowledge on different Operating Server.
- Idea on Relational database management system (RDBMS)
- Ready to face chalanges and solve the Problems quickly.

The Role of DBA is as shown below -



Good

10/6/22

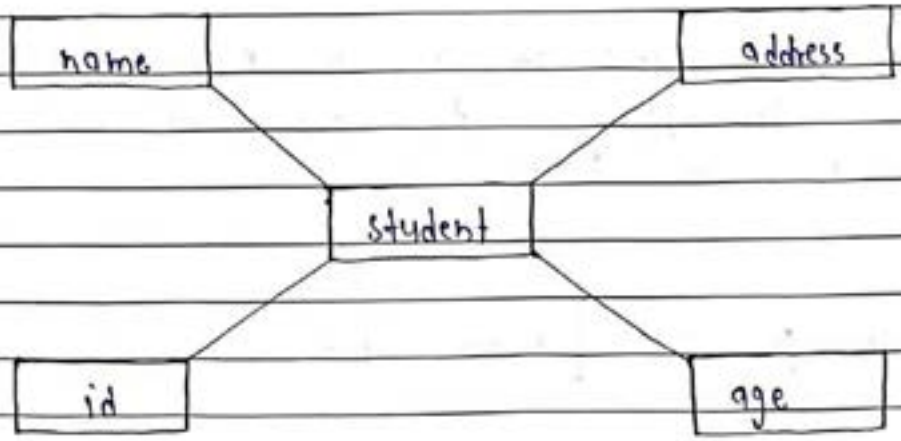
ER Model :-

• ER Model stands for an Entity - Relationship Model. It is high-level data model. This model is used to define the data element and Relationship for a specified system.

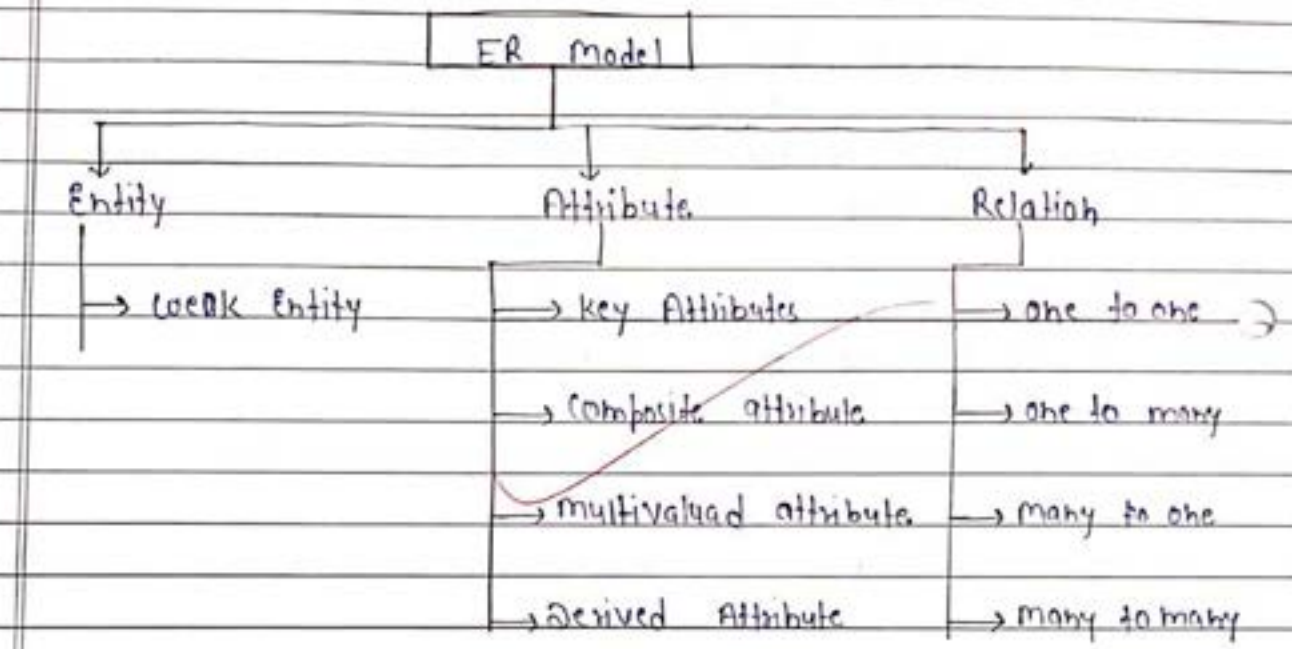
• It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.

• In ER Modeling, the database structure is portrayed as a diagram called an Entity - Relationship diagram.

For example, Suppose we design a school database. In this database, the student will be an Entity with attributes like address, name, id, age etc. the address can be another Entity with attributes like city, street name, Pin code etc. and there will be a Relationship between them.



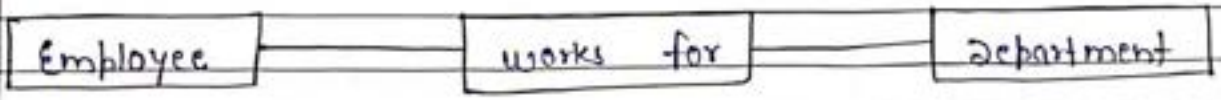
Component of ER Diagram :-



1 Entity :-

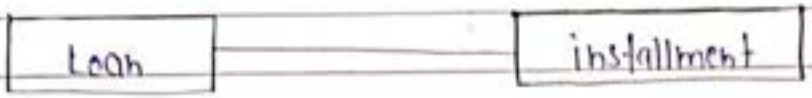
An Entity may be any object, class, person or Place. In the ER diagram, an Entity can be Represented as Rectangles.

Consider an organization as an Example - manager, Product, Employee, department Etc. can be taken as an Entity.



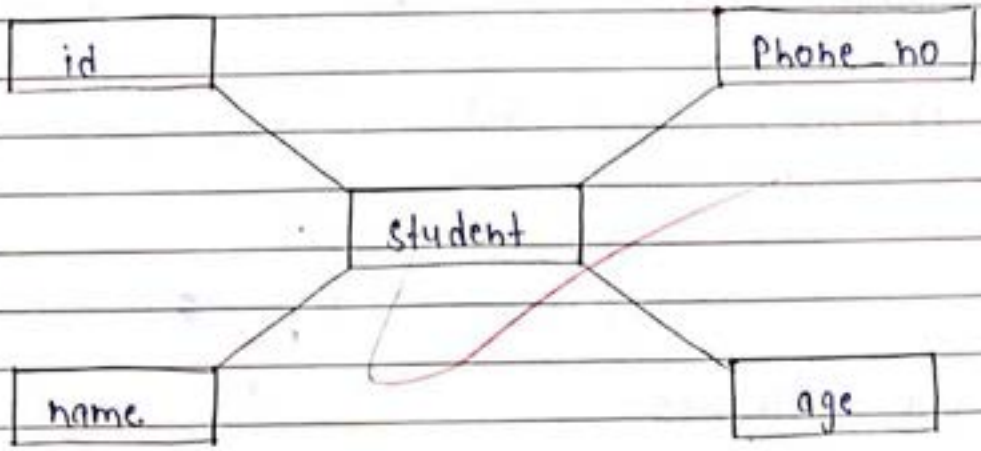
a. Weak Entity :-

An Entity that depends on another Entity called a week Entity. the week Entity doesn't contain any key attributes of its own. the weak Entity is Represented by a double Rectangle.



2. Attribute :-

The attribute is used to describe the Property of an Entity. Eclipse is used to Represent an attribute. for Example, id, age, Contact Number, name, Etc. can be attributes of a student.

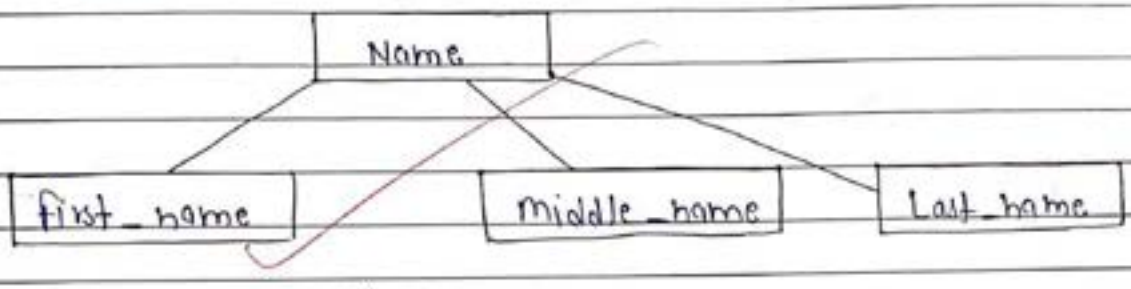


a. Key attribute :-

The key attribute is used to Represent the main characteristics of an Entity. It Represents a primary key. the key attribute is Represented by an Ellipse with the text Underline.

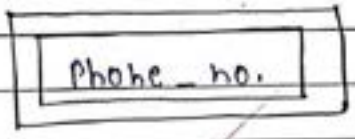
b. Composite Attribute :-

An attribute that composed of many other attribute is known as a composite attribute. the composite attributes is Represented by an Ellipse are Connected with an ellipse.



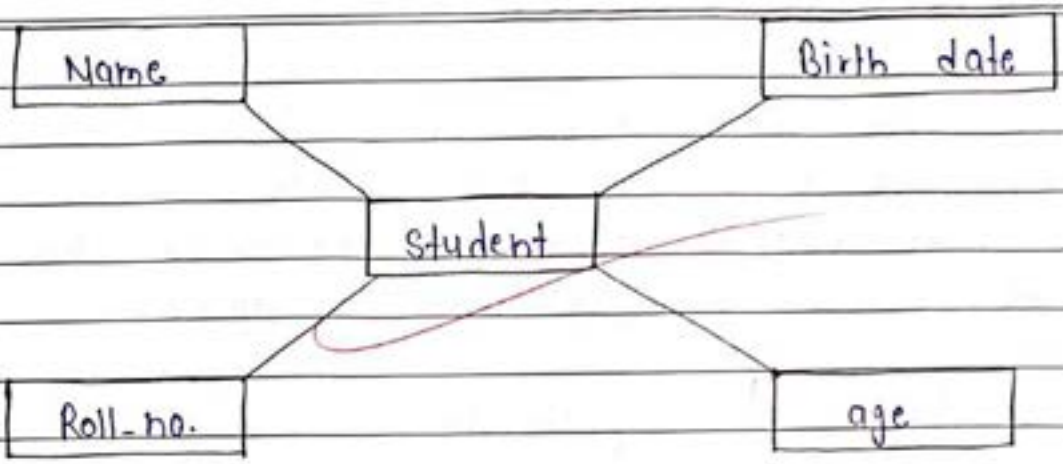
c. Multivalued Attribute :-

An attribute can have more than one value. these attributes are known as a multivalued attribute. the double oval is used to Represent multi-valued attribute.
for example, a student can have more than one phone number.



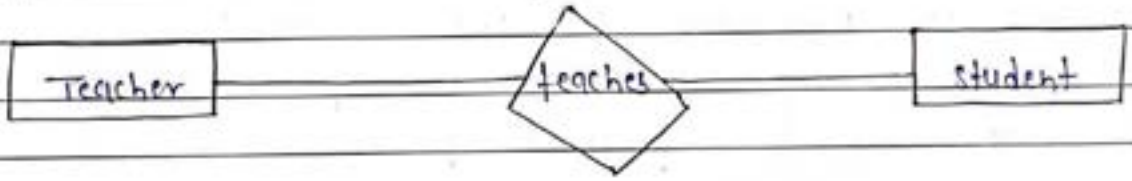
d. Derived Attribute :-

An Attribute that can be derived from other attribute is known as a derived attribute. it can be Represented by a dashed ellipse.
for Example, A Person's age change over time and can be derived from another attribute like Date of birth.



3. Relationship :-

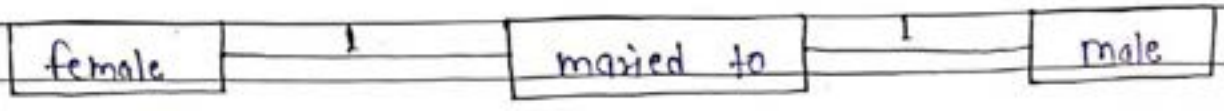
A Relationship is used to describe the Relation between Entities. Diamond or rhombus is used to Represent the Relationship.



a. One - to - One Relationship :-

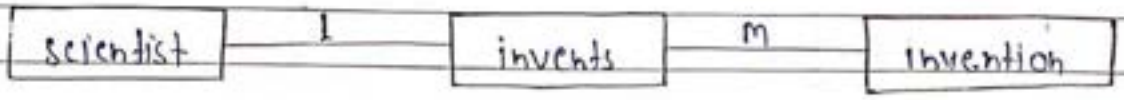
when only one instance of an Entity is associated with the Relationship, then it is known as one to one Relationship.

for Example, A female can marry to one male, and a male can marry to one female.



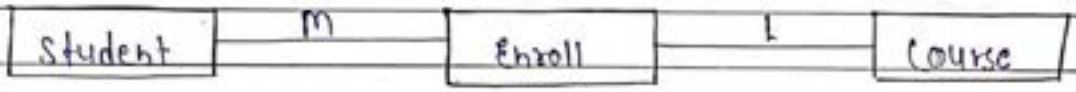
b. One - to - many Relationship :-

When only one instance of the Entity one of the left, and more than one instance of an Entity of the right associates with the Relationship then this is known as a one - to - many Relationship.



c. Many - to - one Relationship :-

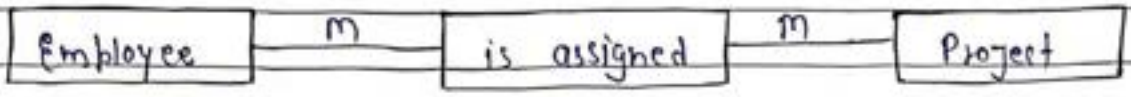
when more than one instance of the Entity of the left, and only one instance of an Entity of the right associates with the Relationship then it is known as a many - to - one Relationship.



d. Many - to - many Relationship :-

when more than one instance of the Entity of the left, and more than one instance of an Entity of the Right associates with the Relationship then it is known as many - to - many Relationship.

for example, Employee can assign be many Projects and Project can have many Employee.



Database Schema :-

- A database schema is the logical Representation. It can store the simplest data, such as a list which shows the data is stored logically in the Entire database. It contains list of attributes and instruction that informs the database Engine that how the data is organized and how the Elements are Related to Each other.

- A database schema contains schema objects that may include tables, fields, Packages, views, Relationships, Primary key, foreign key.

- the shema does not Physically contain the data itself; instead, it gives information about the shape of data and how it can be Related to other tables or models.

- A database schema object includes the following:

  - Consistent formatting for all data Entries.
  - Database objects and unique keys for all data Entries.
  - Tables with multiple columns, and Each column contain its name and datatype.

- the Complexity & the size of the shema vary as Per the size of the Project. It helps developers to Easily manage and structure the database before Coding it.

Types of Database schema :-

the database schema is divided into three types, which are -

1. Logical Schema
2. Physical Schema
3. view Schema

1. Logical Schema :-

A Logical database schema specifies all the logical constraints that need to apply to the stored data. It defines the views, integrity constraints, and tables.

2. Physical database Schema :-

A Physical database schema specifies how the data is stored Physically on a storage system or disk storage in the form of files and indices. designing a database at the Physical level is called a Physical schema.

3. View database Schema :-

The view level design of a database is known as view schema. This schema generally describes the End user interaction with the database systems.

Database Schema Designs :-

A schema design is the first step in building a foundation in data management. Ineffective schema designs are difficult to manage and consume more memory and the Resources.

The list of some Popular database schema designs is given below :

- Flat Model
- Hierarchical Model
- Network Model
- Relational Model
- Star Schema
- Snowflake schema.

Designing Database using Normalization Rules :-

Normalization is a database design technique that Reduces data redundancy and Elimenats undesirable characteristics like insertion, Update and deletion Anomalies. Normalization Rules divides larger table into smaller tables and links them using Relationships. The purpose of Normalization in SQL is to Eliminate Redundant data and Ensure data is stored logically.

The inventor of the Relational model Edgar Codd Proposed the theory of Normalization of data.

Database Normal forms :-

Here is a list of Normal forms in SQL :

- 1NF (First Normal form)
- 2NF (Second Normal form)
- 3NF (Third Normal form)
- BCNF (Boyce - Codd Normal form)
- 4NF (Fourth Normal form)
- 5NF (Fifth Normal form)
- 6NF (Sixth Normal form)

1NF (First Normal form) Rules :-

- Each table cell should contain a single value.
- Each Records needs to be unique.

2NF (Second Normal form) Rules :-

- Rule 1 - Be in 1NF
- Rule 2 - Single column Primary key that does not functionally dependant on any subset of candidate key Relation.

3NF (Third Normal form) Rules :-

- Rule 1 - Be in 2NF
- Rule 2 - Has no transitive functional dependencies.

To move our 2NF table into 3NF, we again need to again devide our table.

BCNF (Boyce - Codd Normal form) :-

Even when a database is in 3rd Normal form, still there would be anomalies resulted if it has more than one Candidate key.

Sometimes is BCNF is also referred as 3.5 Normal forms.

4NF (fourth Normal form) Rules :-

if no database table instance contains two or more, independent and multivalued data describing the Relevant Entity, then it is in 4th Normal form.

5NF (fifth Normal form) Rules :-

A table is in 5th Normal form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.

6NF (Sixth Normal form) Proposed :-

6th Normal form is not standardized, yet however, it is being discussed by database experts for some time.

Hopefully, we would have a clear & standardised definition for 6th Normal form in the near future.

That's all to SQL Normalization !!!

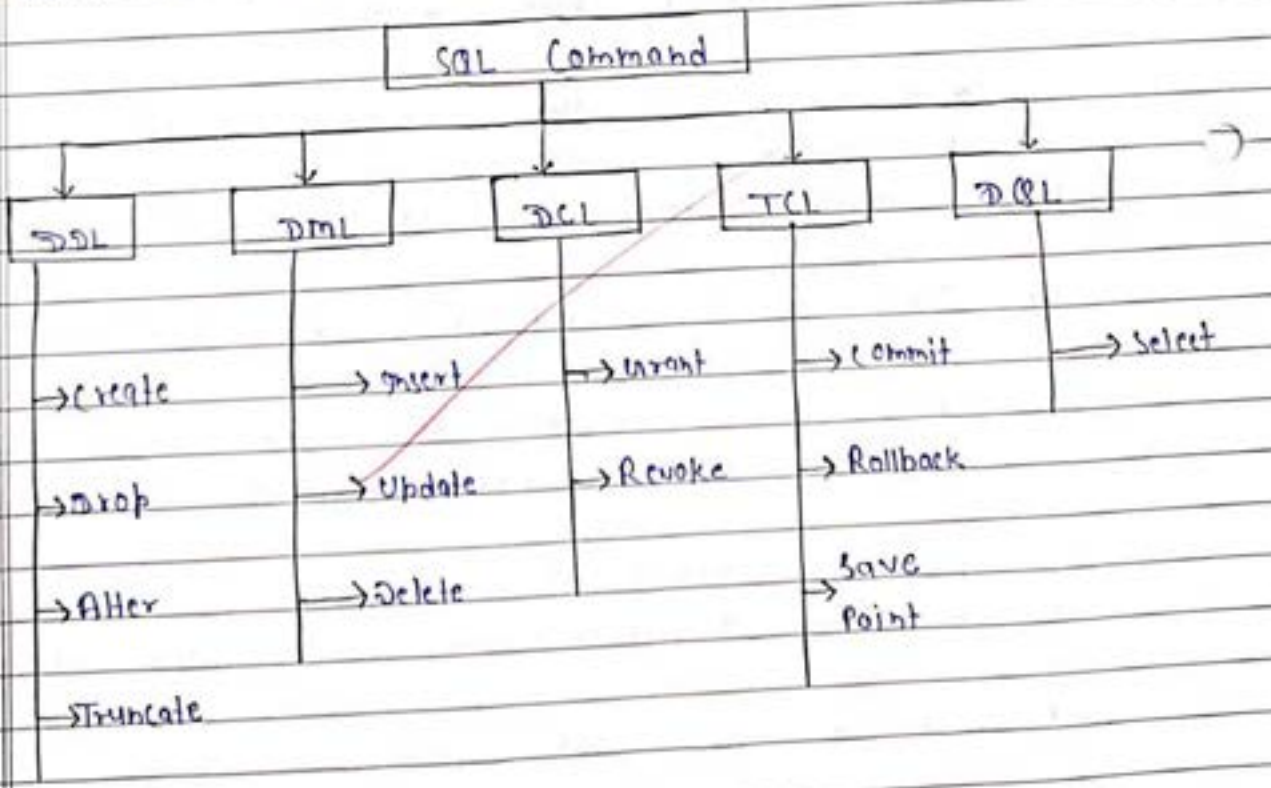
SQL Commands :-

- SQL Commands are instructions. It is used to communicate with the database. It is also used to perform various specific tasks, functions and queries of data.

- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands :-

There are five types of SQL Commands : DDL, DML, DCL, TCL and DQL.



1. Data Definition Language (DDL) :-

- DDL changes the structure of the table like creating a table, deleting a table, Altering a table etc.
- All the command of DDL are auto-commited that means it Permanently Save all the changes in the database.

Here are some commands that come Under DDL :

- Create
- ALTER
- DROP
- TRUNCATE

a. CREATE It is used to create a new table in the database.

Syntax-

```
CREATE TABLE TABLE_NAME ( COLUMN_NAME DATATYPES [...] );
```

Example-

```
CREATE TABLE EMPLOYEE ( NAME VARCHAR2 (20), Email VARCHAR2 (100), DOB DATE );
```

b. DROP: It is Used to delete both the structure and Record Stored in the table.

Syntax-

```
DROP TABLE table_name;
```

Example-

```
DROP TABLE EMPLOYEE;
```

c. ALTER: It is used to Alter the structure of the database. This changed could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax-

```
ALTER TABLE table_name ADD (column_name
                COLUMN - definition;
```

To modify Existing Column in the table:

```
ALTER TABLE table_name MODIFY (Column_definitions....);
```

Example-

```
ALTER TABLE STU_DETAILS ADD ( ADDRESS VARCHAR2(20));
ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));
```

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax-

```
TRUNCATE TABLE table_name;
```

Example-

```
TRUNCATE TABLE EMPLOYEE;
```

2. Data Manipulation Language :- (DML) :-

- DML commands are used to modify the database. It is Responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: the Insert statement is a SQL query. It is used to insert data into the row of table.

Syntax -

```
INSERT INTO TABLE_NAME
(col1, col2, col3, .... colN)
VALUES (value1, value2, value3, .... valueN);
```

Example-

```
INSERT INTO Javapoint (Author, Subject) VALUES ("Sonoo", "DBMS");
```

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax-

```
UPDATE table_name SET [column_name1 = value1,
..... column_nameN = valueN] [WHERE CONDITION];
```

Example -

```
UPDATE Students
SET User_name = "Sonoo"
WHERE Student_Id = '3'
```

c. DELETE: It is used to remove one or more row from a table.

Syntax -

```
DELETE FROM table_name [WHERE CONDITION];
```

Example-

```
DELETE FROM javapoint
WHERE Author = "Sohoo";
```

3. DATA Control Language (DCL) :-

DCL Commands are used to grant and take back authority from any database user.
Here are some commands that come UNDER DCL.

- Grant
- Revoke

a. GRANT : It is used to give user access privileges to a database.

Example-

```
GRANT SELECT, UPDATE ON MY_TABLE TO
SOME_USER, ANOTHER_USER;
```

b. REVOKE : It is used to take back permissions from the user.

Example-

```
REVOKE SELECT, UPDATE ON MY_TABLE
FROM USER1, USER2;
```

10/06/20

Enforcing Primary key and foreign key -

Primary keys :-

A Primary key is a Column or a set of Columns in a table whose values Uniquely identify a row in the table. A Relational database is designed to Enforce the Uniqueness of Primary keys by allowing only one row with a given Primary key value in a table.

Syntax-

```
CREATE TABLE tableName (
    col1 int NOT NULL,
    col2 varchar (50) NOT NULL,
    col3 int,
    .....,
    Primary KEY (col1)
);
```

foreign keys :-

A foreign key is a Column or a set of Columns is a table whose values correspond to the values of the Primary key is another table. In order to add a row with a given foreign key value, there must Exists a row in the Related table with the same Primary key value.

Syntax -

```
CREATE TABLE childTABLE (
col1 int NOT NULL,
col2 int NOT NULL,
col3 int,
.....
PRIMARY KEY (col1),
FOREIGN KEY (col3) REFERENCES ParentTable
(Parent_Primary_key)
);
```

Diagram a Primary and foreign key -

Student

Stud_Id (Primary key)	Name	Course
101	John	Computer
105	Mary	AI
107	Sheero	Biology
108	Bisle	Maths

Department

Dept_name	Stud_Id (Foreign key)
CS_department	105
CS_department	101
Science_department	107
Maths_department	108

SQL CREATE INDEX Statement :-

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 user cannot see the indexes, they are just used to speed up searches / queries.

CREATE INDEX Syntax :-

Creates an index on a table. Duplicate values are allowed.

```
CREATE INDEX index_name
ON table_name (column1, column2, ....);
```

CREATE UNIQUE INDEX Syntax :-

Creates a Unique index on a table. Duplicate values are not Allowed.

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ....);
```

Note - The Syntax for Creating indexes varies among different databases. Therefore : check the Syntax for Creating indexes in your database.

CREATE INDEX Example :- the SQL Statement below creates an index named "idx_lastname" on the "LastName" columns in the "Persons" table:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

If you want to create an index on a combination of colums.

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

DROP INDEX Statement :- The DROP INDEX Statement is used to delete an index in a table.

MS Access :

```
DROP INDEX index_name ON table_name;
```

SQL Server :

```
DROP INDEX table_name.index_name;
```

DB2 / Oracle :

```
DROP INDEX index_name;
```

MySQL :

```
ALTER TABLE table_name
DROP INDEX index_name;
```

Concepts of transaction :-

- The transaction is a set of logically Related operation. It Contains a group of tasks.

- A transaction is an action or series of actions. It is Performed by a Single user to Perform operation for accessing the Contends of the database.

Example- Suppose an Employee of bank transfers Rs 800 from X's account to Y's account. This Small transaction Contains several low - level tasks :

X's Account :

```
Open_account (X)
Old_Balance = X.balance
New_Balance = old_Balance - 800
X.balance = New_Balance
Close_Account (X)
```

Y's Account :

```
Open_account (Y)
Old_Balance = Y.balance
New_Balance = old_Balance + 800
Y.balance = New_Balance
Close_Account (Y)
```

Operations of transaction :-

following are the main operations of transaction :

Read (X) : Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

write (X) : write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations.

1. R (X);
2. X = X - 500;
3. W (X);

Transaction Property :-

The transaction has the four properties. These are used to maintain consistancy in a database, before and After the transaction.

Property of transaction :-

1. Atomicity
2. Consistency
3. Isolation
4. Durability

1. Atomicity :-

- It states that all operations of the transaction take place at once if not, the transaction is aborted.

• There is no midway, i.e the transaction can not occur partially. Each transaction is treated as one unit the either run to completion or is not executed to all.

2. Consistency :-

• The integrity constraints are maintained so that the database is consistant before and after the transaction.

• The execution of a transaction will leave a database in either its prior stable state or a new stable state.

• The consistance Poperty of database states that every transaction sees a consistent database instance.

• The transaction is used to transform the database from one consistent state to another consistent state.

For Example : The total ammount must be maintained before or after the transaction.

Total before T occurs = 600 + 300 = 900
Total After T occurs = 500 + 400 = 900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occure.

3. Isolation :-

- It shows that the data which is used at the time of execution of a transaction can not be used by the second transaction until the first one is completed.

- In isolation, if the transaction T1 is being executed and using the data item x, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.

- The Concurrency Control Subsystem of the DBMS Enforced the isolation Property.

4. Durability :-

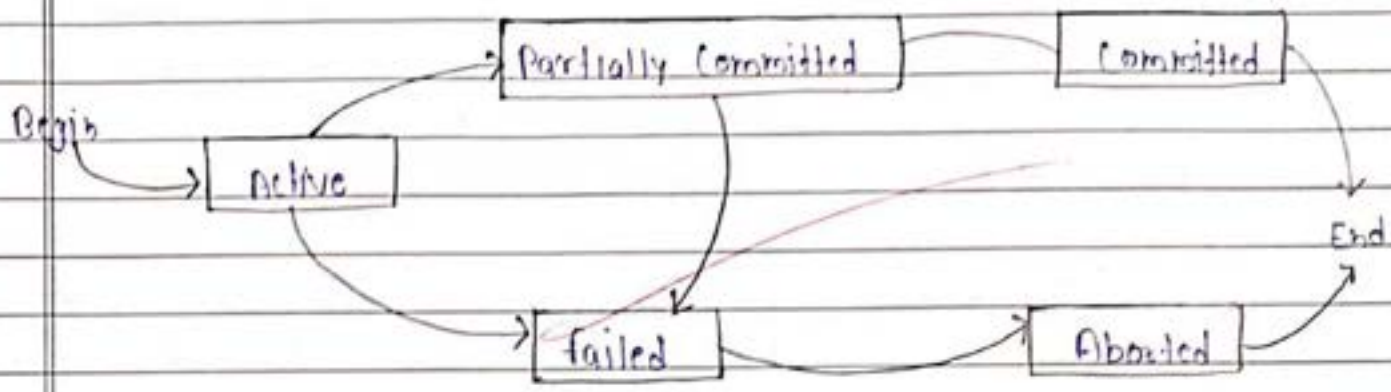
- The durability property is used to indicate the the performance of the databases consistant state. it states that the transaction made the permanent changes.

- They can not be lost by the erroneous operation of a faulty transaction or by the system failure.

- The Recovery Subsystem of the DBMS has the Responsibility of Durability Property.

States of Transaction :-

In a database, the transaction can be in one of the following states.



Active state :-

- The active state is the first state of every transaction is being executed.
- for Example : insertion of deletion or updating a Record is done here. But all the Records are still not saved to the database.

failed state :-

- if any of the checks fail and the transaction has reached a failed state then the database Recovery system will make sure that the database is in its Previous Consistent state. if not then it will abort or roll back the transaction to bring the database into a Consistent state.

ACID Properties in DBMS :-

A transaction is a single logical unit of work that accesses and possibly modifies the contains of a database. Transactions access data using Read and write operations. In order to maintain consistency in a database, before and after the transaction, certain Properties are followed. These are called ACID Properties.

A (Atomicity) - The Entire transaction takes Place at once or does not happen at all.

C (Consistency) - The database must be Consistent before and after the transaction.

I (Isolation) - Multiple transactions occur independently without interferences.

D (Durability) - The changes of a successful transaction occurs even if the system failure occurs.

Abort : if a transaction aborts, changes made to the database are not visible.

Commit : if a transaction commits, changes made are visible.

Atomicity is also known as the 'All or Nothing Rule'.

SQL Joins :-

A JOIN clause is used to combine rows from two or more tables, based on a Related column between them.

Let's Look at a Selection from the Orders tables -

Order Id	Customer ID	Orderdate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a Selection from the "Customers" table-

CustomerID	CustomerName	ContactName	Country
1	kuldeep kumar	kuldeep	India
2	SATYAM SHARMA	SATYAM	mexico
3	DIWAKAR VERMA	DIWAKAR	mexico

that the "CustomerID" Column in the "Orders" table Refers to the "CustomerID" in the "Customers" table. The Relationship between the two tables above in the "CustomerID" Column.

Then, we can create the following SQL Statement (that Contains an INNER JOIN).

Example-

```
SELECT Orders.OrderID, Customers.CustomerName,
Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID =
Customers.CustomerID;
```

Different Types of SQL JOINs :-

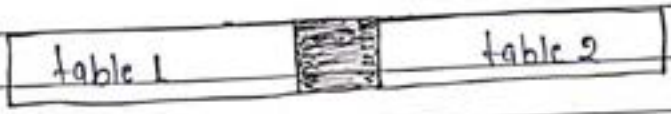
- (INNER) JOIN
- LEFT (OUTER) JOIN
- RIGHT (OUTER) JOIN
- FULL (OUTER) JOIN

SQL INNER JOIN keyword :-

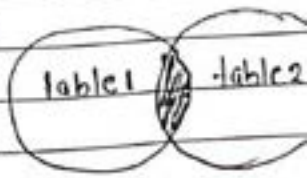
The INNER JOIN keyword Selects Records that have matching values in both tables.

Syntax-

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



Or



Atomicity is also known as the 'All or Nothing Rule'.

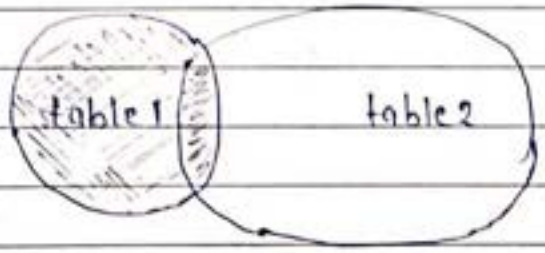
SQL JOIN LEFT keyword :-

The LEFT JOIN keyword Returns all Records from the left table (table 1), and the matching Records from the Right table (table 2). 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;
```

Note - In some databases LEFT JOIN is called LEFT OUTER JOIN.



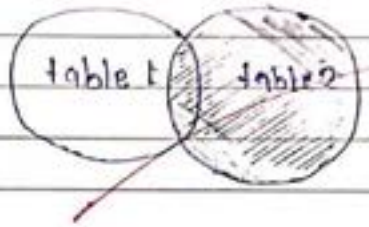
SQL RIGHT JOIN Keyword :-

The RIGHT JOIN keyword Returns all Record from the Right table (table 2), and the matching Records from the left table (table 1). The result is 0 Records from the Left Side, if there is no match.

Syntax-

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

Note- In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



SQL FULL OUTER JOIN keyword :-

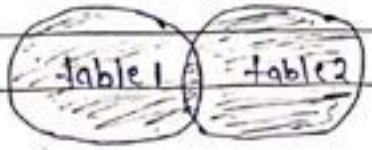
The FULL OUTER JOIN keyword returns all Records when there is a match in left (table1) or right (table2) table records.

Tip- 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;
```

Note- FULL OUTER JOIN can Potentially Return very large Result - Sets !



Indexing in DBMS :-

- Indexing in used to optimize the performance of a database by minimizing the number of disk accesses Required when a query is processed.

- The index is a type of data structure. It is used to located and access the data in a database table quikly.

Index Structure :-

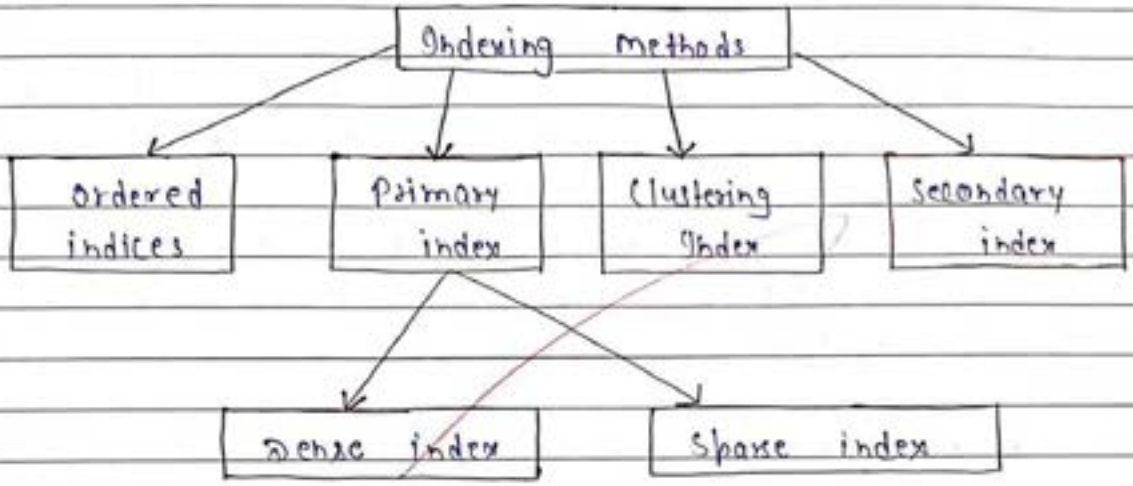
Index can be Created using some database Columns.

search key	Data Reference

- The first Column of the database is the Search key that Contains a Copy of the primary key or Candidate key of the table. The values of the primary key are Stored in sorted order so that the Corresponding data can be accessed easily.

- The second Column of the database is the data Reference. It Contain a set of pointers holding the address of the disk block where the value of the particular key can be found.

Indexing methods :-



Ordered indices :-

the indices are usually sorted to make searching faster. the indices which are sorted are known as ordered indices.

Example. Supose we have an Employee table with thousands of Record and each of which is 10 bytes long.

- In the case of database with no index, we have to search the disk block from starting till it Reaches 543.

- In the case of an index, we will search using Indexes and the DBMS will Read the Record after Reading 542 * 2 = 1084 bytes.

Primary Index :-

- If the Index is Created on the basis of the primary key of the table, then it is known as primary Indexing. These primary keys are Unique to Each Record and Contain 1:1 Relation between the Records.

- As Primary keys are stored in Sorted order, the Performance of the Searching Operation is quite efficient.

- The Primary index can be classified into two types.

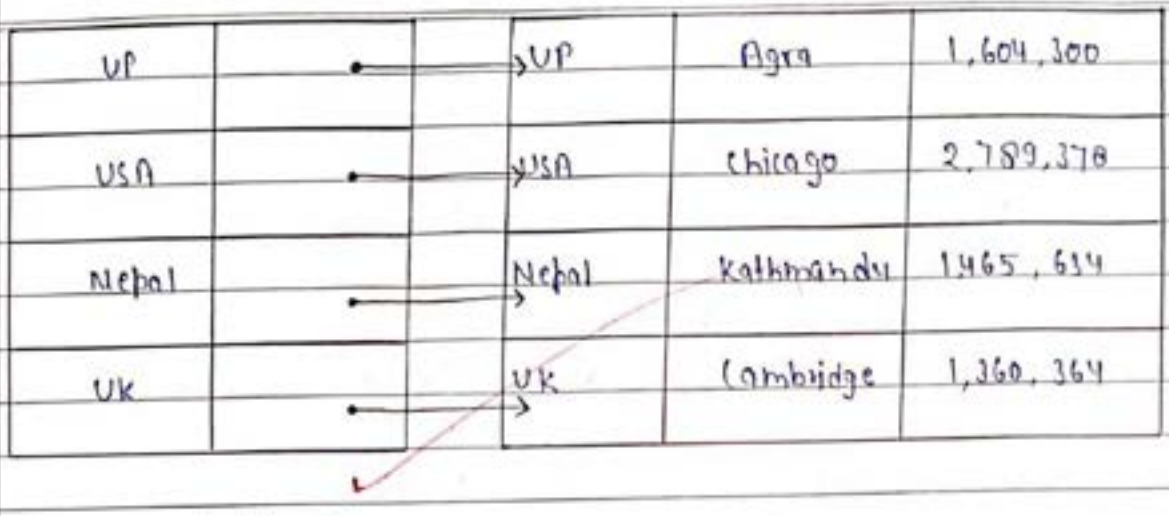
1. Dense index
2. Sparse index

1. Dense Index :-

- The dense index Contains an index Record for every search key value in the data file. it makes Searching faster.

- In this, the number of Records in the index table is same as the number of Records in the main table.

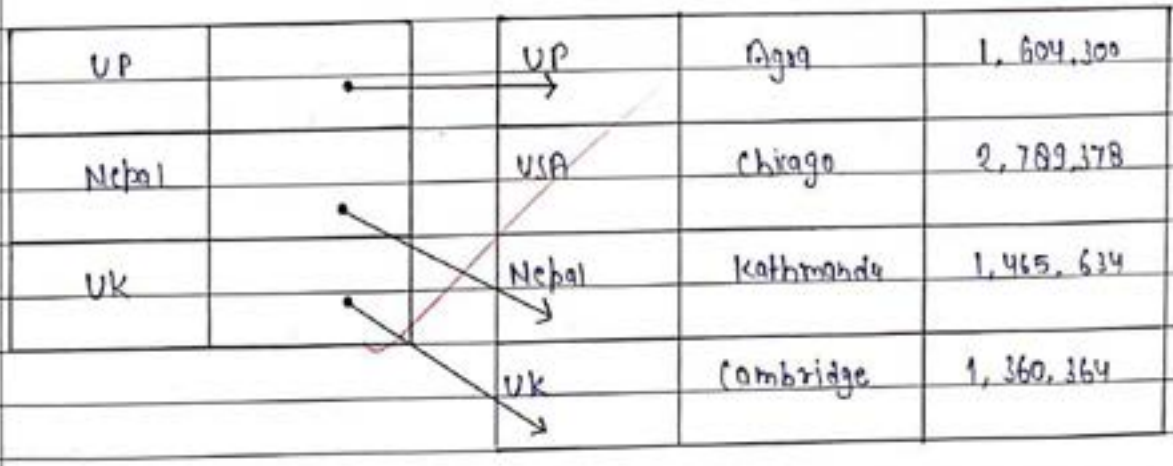
- It needs more space to store index Record itself. The index Records have the Search key and a Pointer to the actual Record on the disk.



2. Sparse index :-

- In the data file, index Record appears only for a few items. Each item Points to a block.

- In this, instead of Pointing to each Record in the main table, the index Points to the Records in the main table in a gap.



Clustering Index :-

- The Recods with have similar characteristic are grouped, and indexes are created for these group.

• A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be Unique for Each Record.

• In this case, to identify the Record faster, we will group two or more columns to get the Unique value and create index out of them. This method is called a clustering index.

Secondary Index :-

In the sparse indexing, as the size of the table grows the size of mapping also grows. These mappings are usually kept in the Primary memory so that address fetch should be faster. The the secondary memory searches the actual database on the address got from mapping. If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

In secondary indexing, to Reduce the size of mapping, another level of indexing is introduced. In this method, the huge Range for the columns is Selected initially so that the mapping size of the first level becomes small. Then Each Range is further divided into smaller Ranges. the mapping of the first level is Stored in the Primary memory, so that address fetch is faster.

Stored Procedure :-

A stored Procedure is a prepared SQL code that you can save, so the 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 the Stored Procedure can act based on the Parameter value (s) that is Passed.

Stored Procedure Syntax :-

```
CREATE PROCEDURE Procedure_name
AS
sql_statement
GO;
```

Execute a Stored Procedure :-

```
EXEC Procedure_name;
```

Stored Procedure to Example :-

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

Stored Procedure with one Parameter :-

The following SQL statement creates a stored procedure that selects customers from a particular city from the "Customers" table :

Example-

```
CREATE PROCEDURE SelectAllCustomers
@City nvarchar (30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

Stored Procedure with multiple Parameters :-

Setting up multiple Parameter is very easy. just list each parameter and the datatype seberated by a comma as show below.

The following SQL statement creats a stored procedure that Select Customers from a particular city with a particular postal code from the "Customers" table :

Example-

```
CREATE PROCEDURE SelectAllCustomers
@City nvarchar (30), @PostalCode nvarchar(10)
AS
SELECT * FROM Customers WHERE City =
@City AND PostalCode = @PostalCode
GO;
```

Cursor in SQL :-

- A Machanism to navigate tuple - by - tuple over a Relation.
- Typically used inside triggers, stored Procedures.
- when we Execute a Query, a Relation is Returned
- It is stored in Private work area for the Query.
- Cursor is a Pointer to this area

- Move the cursor to Navigate over the tuples.
- Enables users to loop around a Slection of data.
- use Complex action which would not be feasible in Standard SQL Selection Queries.

Syntax for Cursors :-

- Declare as a variable in the same way as standard variables.

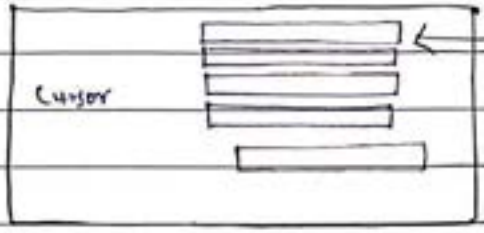
- Identified as Cursor type
SQL included

Eg.

```
Cursor Cur_Emp is
Select Emp_id, Surname name, grade, Salary
from Employee
where grade is true;
```

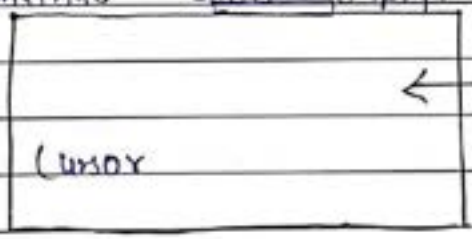
Controlling Cursor :-

Open the Cursor.

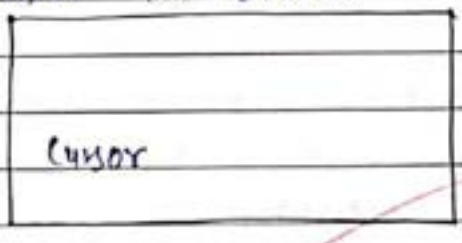


Fetch a row from the cursor.

Continue Until Empty.



Close the Cursor.



Example of Cursor :-

```
DECLARE @CustomerId INT

DECLARE Cursor1 CURSOR READ_ONLY
FOR
SELECT CustomerId
```

```
FROM Customers

OPEN Cursor1

FETCH NEXT FROM Cursor1 INTO
@CustomerId

WHILE @@FETCH_STATUS = 0
BEGIN
        PRINT @CustomerId

FETCH NEXT FROM Cursor1 INTO
@CustomerId
END

        CLOSE Cursor1
        DEALLOCATE Cursor1
```

Advantage of Cursor :

- Cursor are best used when Performing row-by-row Operations that can't be accomplished with set-based Operations.
- Quick and dirty.

Disadvantage of Cursor :

- A factor affecting Cursor Speed is the Number of rows and Columns brought into the Cursor. Time how long it takes to Open your Cursor and fetch Statements.
- Speed and Performance issues.

Cursor Example in mySQL :-

```
DELIMITER $
DROP PROCEDURE IF EXISTS ShowCourseLecturesAlt $
CREATE PROLEDURE showCourseLecturesAlt ( IN CourseId INT)
BEGIN
  DECLARE lectSubject VARCHAR (128) ;
  DECLARE lectNum INT (2) ;
  DECLARE finishedFlag INT ;
  DECLARE lectCursor CURSOR FOR
  SELECT Subject , num_lecture FROM lecture WHERE
  Course_lecture = CourseId ;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET
  finishedFlag = 1 ;
  OPEN lectCursor ;
  SET finished flag = 0 ;
  REPEAT
  FETCH lectCursor INFO lectSubject , lectNum ;
  IF ( finishedFlag = 0 ) THEN
      SELECT lectNum AS '.....' , lectSubject AS
                                  '.....' ;
  END IF ;
  UNTIL ( finishedFlag = 1 )
  END REPEAT ;
  CLOSE lectCursor ;
  END $
  DELIMITER ;
```

Triggers :-

- Triggers are very similar to stored procedures one big difference is :
Trigger can not be manually Executed.

- They only execute in Response to a user Action, like an Insert is called when an Events occures in the table.

- Are used to : check the validity of the data that are Inserted into a table.

- To Calculate derived values, to maintain the Sign in logs and the insertions in a table. Implementation Cost -> time Consuming.

Basic Commands :-

- Creation
- CREATE TRIGGER
- DELETE
- DROP TRIGGER < trigger Name >
- SHOW TRIGGER Code
- SHOW CREATE TRIGGER < trigger name >
- Show list of Created triggers
- SHOW TRIGGER
- Call / Execute a trigger
- We Cannot Call a trigger, like a Procedure
- It is Executed when an Event happens Call it.

Designing a trigger :-

- To design a trigger we have to determine
- Th which table it would be applied
- With what event it will be linked
- Eg INSERT, UPDATE, DELETE
- When it will be executed
- Before the Event
- After the Event
- Its functionality
- At the main body of the trigger we can write SQL Code.

Trigger Creation :-

```
CREATE TRIGGER trigger_name trigger_time trigger_event
ON Table_name
FOR EACH ROW trigger_body
```

- trigger_name
- trigger_time -> when it will be executed (after or before)
- trigger_event -> the Connected Event
- ON table_Name -> the table is belongs to
- FOR EACH ROW -> It will be executed for each row.
- trigger_body -> the SQL Scripts

Trigger Events :-

- The Triggers Events could be
- INSERT
- UPDATE
- DELETE
- TRUNCATE & DROP don't call a trigger.
- We could have at max 6 triggers in every table.

Good