Data:

Facts, figures, statistics etc. having no particular meaning (e.g. 1, ABC, 19 etc).

Database :-

is a collection of related data and data is a collection of facts and figures that can be processed to produce information.

Mostly data represents recordable facts. Data aids in producing information, which is based on facts. For example, if we have data about marks obtained by all students, we can then conclude about toppers and average marks.

A database management system stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

Database Management System or DBMS

in short refers to the technology of storing and retrieving users' data with utmost efficiency along with appropriate security measures. This tutorial explains the basics of DBMS such as its architecture, data models, data schemas, data independence, E-R model, relation model, relational database design, and storage and file structure and much more.

Characteristics

Traditionally, data was organized in file formats. DBMS was a new concept then, and all the research was done to make it overcome the deficiencies in traditional style of data management. A modern DBMS has the following characteristics

1. **Real-world entity**
   A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use students as an entity and their age as an attribute.

2. **Relation-based tables**
   DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the table names.

3. **Isolation of data and application**
   A database system is entirely different than its data. A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about data, to ease its own process.

4. **Less redundancy**

   DBMS follows the rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Normalization is a mathematically rich and scientific process that reduces data redundancy.

5. **Consistency**

   Consistency is a state where every relation in a database remains consistent. There exist methods and techniques, which can detect attempt of leaving database in inconsistent state. A DBMS can provide greater consistency as compared to earlier forms of data storing applications like file-processing systems.

6. **Query Language**

   DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and as different filtering options as required to retrieve a set of data. Traditionally it was not possible where file-processing system was used.

7. **ACID Properties**

   DBMS follows the concepts of **A**tomicity, **C**onsistency, **I**solation, and **D**urability (normally shortened as ACID). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.

8. **Multiuser and Concurrent Access**

   DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.

9. **Multiple views**

   DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.

10. **Security**

Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features. For example, a user in the Sales department cannot see the data that belongs to the Purchase department. Additionally, it can also be managed how much data of the Sales department should be displayed to the user. Since a DBMS is not saved on the disk as traditional file systems, it is very hard for miscreants to break the code.

ADVANTAGES OF A DBMS

1- Database Development:

It allows organizations to place control of database development in the hands of database administrators (DBAs) and other specialists.

2-Data independence:

Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

3-Efficient data access:

A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. It allows different user application programs to easily access the same database. Instead of having to write computer programs to extract information, user can ask simple questions in a query language.

4-Data integrity and security:

If data is always accessed through the DBMS, the DBMS can enforce :

• integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded.

• Also, the DBMS can enforce access controls that govern what data is visible to different classes of users.

5-Crash recovery:
the DBMS protects users from the effects of system failures.

6- Data administration and Concurrent access:
When several users share the data(more than one user access the database at the same time), DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time

Users

A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows

1. **Administrators**
   Administrators maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used. They create access profiles for users and apply limitations to maintain isolation and force security. Administrators also look after DBMS resources like system license, required tools, and other software and hardware related maintenance.

2. **Designers**
   Designers are the group of people who actually work on the designing part of the database. They keep a close watch on what data should be kept and in what format. They identify and design the whole set of entities, relations, constraints, and views.

3. **End Users**
   End users are those who actually reap the benefits of having a DBMS. End users can range from simple viewers who pay attention to the logs or market rates to sophisticated users such as business analysts.

DBMS – Architecture

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent n modules, which can be independently modified, altered, changed, or replaced.

a) 1-tier architecture

the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.

b) 2-tier architecture

If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming**.**

c) 3-tier Architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.

1. **Database (Data) Tier**

At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.

2. **Application (Middle) Tier**

At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.

3. **User (Presentation) Tier**

End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

Relational model:-

The relational model (RM) for database management is an approach to managing data using a structure and language consistent with first-order predicate logic, first described in 1969 by Edgar F. Codd . In the relational model of a database, all data is represented in terms of tuples, grouped into relations. A database organized in terms of the relational model is a relational database**.**

The purpose of the relational model is to provide a declarative method for specifying data and queries: users directly state what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries.

Relational data model is the primary data model, which is used widely around the world for data storage and processing .This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Concepts:-

Tables :-

In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

Tuple :-

A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance :-

A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema :-

A relation schema describes the relation name (table name), attributes, and their names.

Relation key :-

Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain :-

Every attribute has some pre-defined value scope, known as attribute domain.

Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints −

- Key constraints
- Domain constraints
- Referential integrity constraints

Key Constraints :-

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there are more than one such minimal subsets, these are called ***candidate keys***.

Key constraints force that :-

- in a relation with a key attribute, no two tuples can have identical values for key attributes.
- a key attribute can not have NULL values.

Key constraints are also referred to as Entity Constraints.

Domain Constraints

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a

relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

Referential integrity Constraints

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages − relational algebra and relational calculus.

Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation − $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

$\sigma_{subject = "database"}(Books)$

Output − Selects tuples from books where subject is 'database'.

$\sigma_{subject = "database" \text{ and } price = "450"}(Books)$

Output :- Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$

Output :- Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1, A2, An} (r)$

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

example :-

$\prod_{subject, author} (Books)$

Selects and projects columns named as subject and author from the relation Books.

Union Operation (∪)

It performs binary union between two given relations and is defined as −

$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$

Notion − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

- **r**, and **s** must have the same number of attributes.

- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

Output − Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − **r** − **s**

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

Output − Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

$\sigma_{author = 'tutorialspoint'}$(Books X Articles)

Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** ρ.

Notation − ρ_x (E)

Where the result of expression **E** is saved with name of **x**.

Additional operations are −

- Set intersection
- Assignment
- Natural join

Relational Calculus

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Relational calculus exists in two forms −

Tuple Relational Calculus (TRC)

Filtering variable ranges over tuples

Notation − {T | Condition}

Returns all tuples T that satisfies a condition.

example :-

{ T.name | Author(T) AND T.article = 'database' }

Output − Returns tuples with 'name' from Author who has written article on 'database'.

TRC can be quantified. We can use Existential (∃) and Universal Quantifiers (∀).

example :-

{ R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}

Output − The above query will yield the same result as the previous one.

Domain Relational Calculus (DRC)

In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation −

{ a_1, a_2, a_3, ..., a_n | P (a_1, a_2, a_3, ... ,a_n)}

Where a1, a2 are attributes and **P** stands for formulae built by inner attributes.

example :-

{< article, page, subject > | ∈ TutorialsPoint ∧ subject = 'database'}

Output − Yields Article, Page, and Subject from the relation TutorialsPoint, where subject is database.

Just like TRC, DRC can also be written using existential and universal quantifiers. DRC also involves relational operators.

The expression power of Tuple Relation Calculus and Domain Relation Calculus is equivalent to Relational Algebra.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

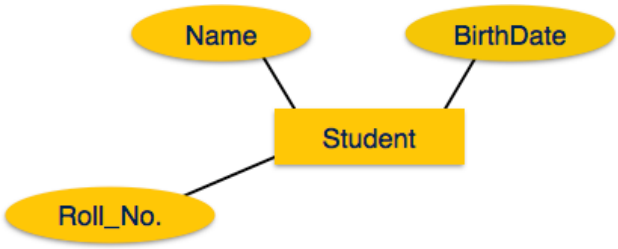
Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

Attributes

Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

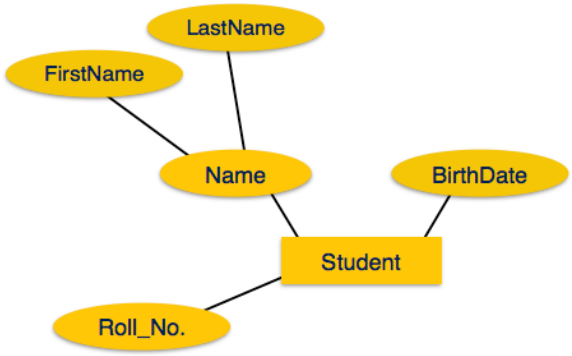
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



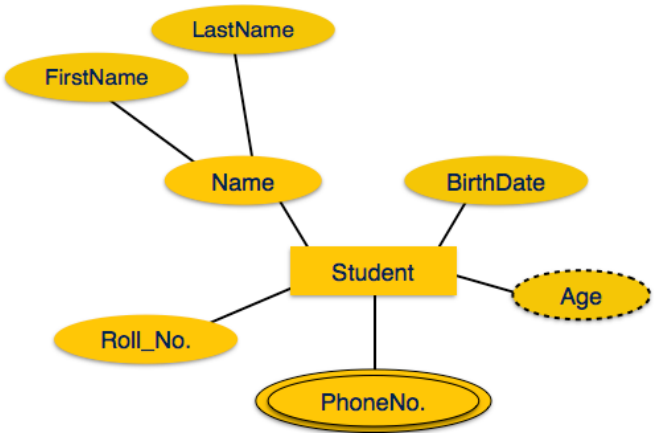
Types of Attributes

- **Simple attribute** − Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute** − Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.

  If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.

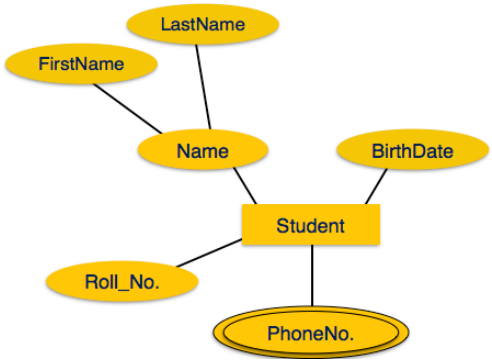


- **Derived attribute** − Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth. **Derived** attributes are depicted by dashed ellipse.



- **Single-value attribute** − Single-value attributes contain single value. For example − Social_Security_Number.
- **Multi-value attribute** − Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

Multivalued attributes are depicted by double ellipse.



These attribute types can come together in a way like −

- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes

- composite multi-valued attributes

Entity-Set and Keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, the roll_number of a student makes him/her identifiable among students.

- **Super Key** − A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key** − A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Primary Key** − A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

Functional Dependency

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

Armstrong's Axioms

If F is a set of functional dependencies then the closure of F, denoted as F^+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

- **Reflexive rule** : If alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.
- **Augmentation rule** : If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- **Transitivity rule** : Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.

Trivial Functional Dependency

- **Trivial** : If a functional dependency (FD) X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.
- **Non-trivial** : If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD.
- **Completely non-trivial** : If an FD X → Y holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD.

Entity-Relationship Model

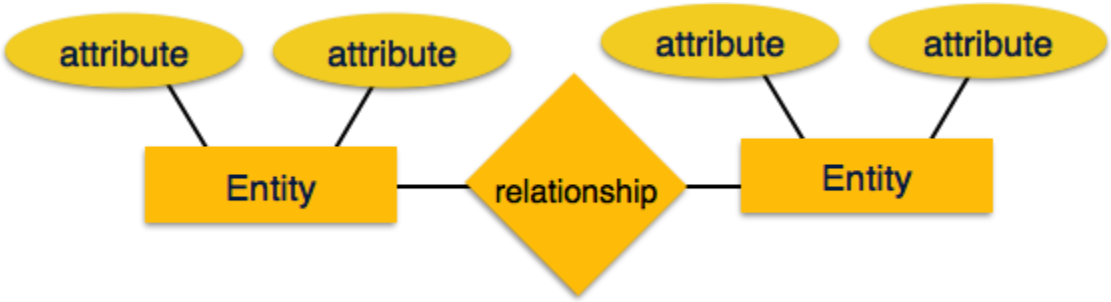
Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of a database.

ER Model is based on :-

- **Entities** and their *attributes.*
- **Relationships** among entities.

These concepts are explained below.



- **Entity** − An entity in an ER Model is a real-world entity having properties called **attributes**. Every **attribute** is defined by its set of values called **domain**. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.

Relationship

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

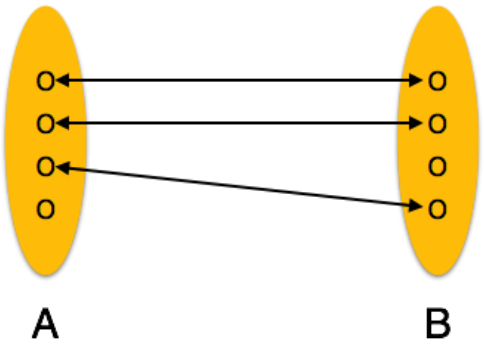
Binary Relationship and Cardinality

A relationship where two entities are participating is called a **binary relationship**.

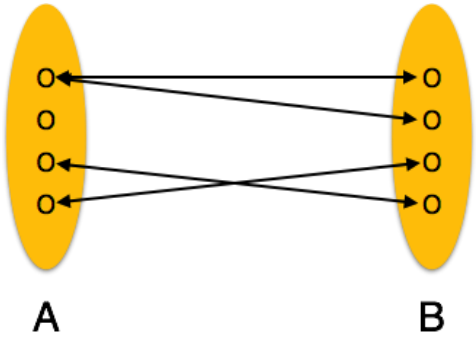
Mapping Cardinalities

Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

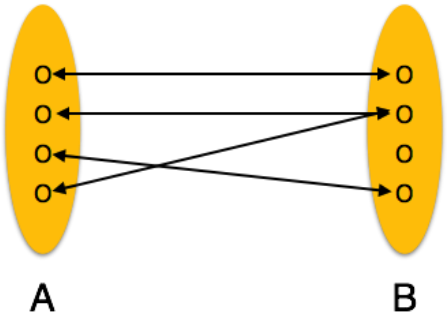
- **One-to-one** : One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



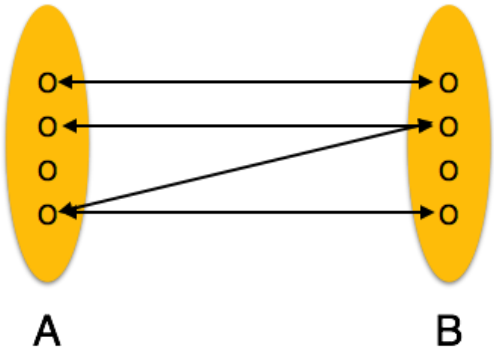
- **One-to-many** − One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



- **Many-to-one** − More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.

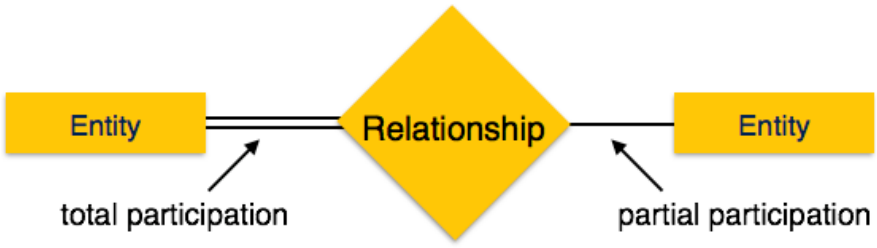


- **Many-to-many** − One entity from A can be associated with more than one entity from B and vice versa.



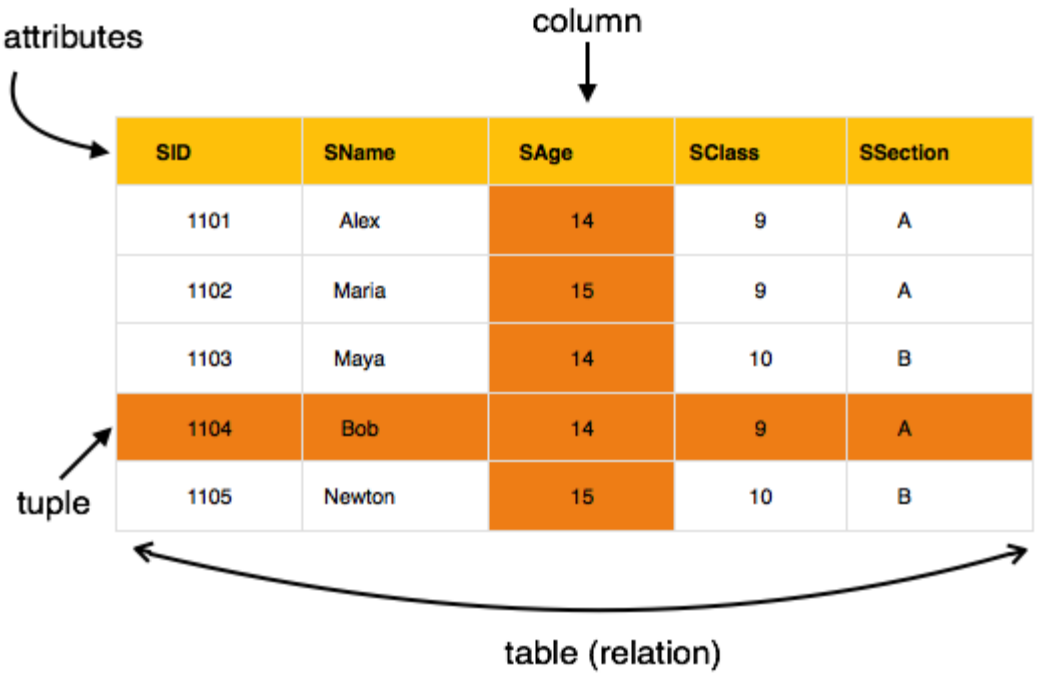
Participation Constraints

- **Total Participation** − Each entity is involved in the relationship. Total participation is represented by double lines.
- **Partial participation** − Not all entities are involved in the relationship. Partial participation is represented by single lines.



Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.



The main highlights of this model are –

- Data is stored in tables called **relations**.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.

Normalization

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **Update anomalies** − If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few

instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

- **Deletion anomalies** − We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies** − We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

First Normal Form

First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

We re-arrange the relation (table) as below, to convert it to First Normal Form.

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

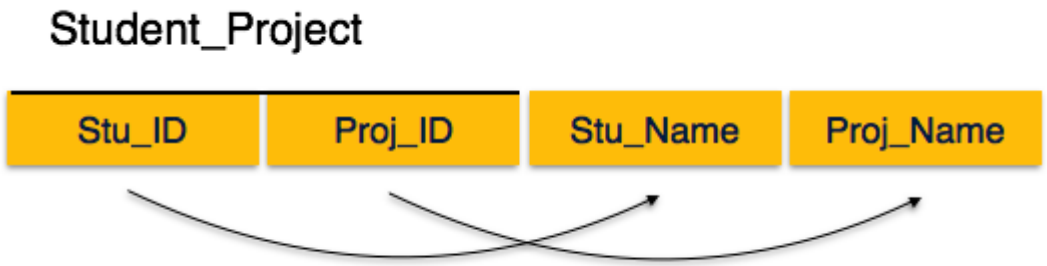
Each attribute must contain only a single value from its pre-defined domain.

Second Normal Form

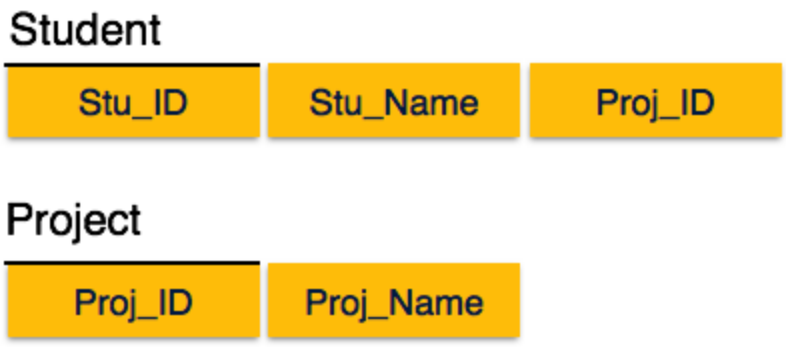
Before we learn about the second normal form, we need to understand the following −

- **Prime attribute** − An attribute, which is a part of the prime-key, is known as a prime attribute.
- **Non-prime attribute** − An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.



We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.



We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

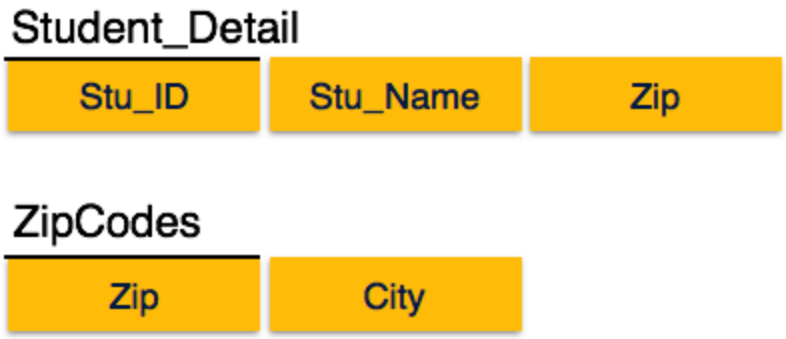
Third Normal Form

For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy −

- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either −
    - X is a superkey or,
    - A is prime attribute.

We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists **transitive dependency**.

To bring this relation into third normal form, we break the relation into two relations as follows −



Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that −

- For any non-trivial functional dependency, X → A, X must be a super-key.

In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,

Stu_ID → Stu_Name, Zip

and

Zip → City

Which confirms that both the relations are in BCNF.

We understand the benefits of taking a Cartesian product of two relations, which gives us all the possible tuples that are paired together. But it might not be feasible for us in certain cases to take a Cartesian product where we encounter huge relations with thousands of tuples having a considerable large number of attributes.

Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied .

We will briefly describe various join types in the following sections.

Theta (θ) Join

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol **θ**.

Notation

$R1 \bowtie_\theta R2$

R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don’t have anything in common, that is $R1 \cap R2 = \Phi$.

Theta join can use all kinds of comparison operators.

Student		
SID	**Name**	**Std**
101	Alex	10
102	Maria	11

Subjects	
Class	**Subject**
10	Math
10	English
11	Music
11	Sports

Student_Detail –

$$\text{STUDENT} \bowtie_{\text{Student.Std = Subject.Class}} \text{SUBJECT}$$

Student_detail				
SID	**Name**	**Std**	**Class**	**Subject**
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Equijoin

When Theta join uses only **equality** comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

Natural Join (⋈)

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Courses

CID	Course	Dept
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

HoD

Dept	Head
CS	Alex
ME	Maya
EE	Mira

Courses ⋈ HoD

Dept	CID	Course	Head
CS	CS01	Database	Alex

ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

Outer Joins

Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins − left outer join, right outer join, and full outer join.

Left Outer Join(R ⟕S)

All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Left	
A	**B**
100	Database
101	Mechanics
102	Electronics

Right	
A	**B**
100	Alex

102	Maya
104	Mira

Courses ⟗ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya

Right Outer Join: (R ⟖ S)

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Courses ⟖ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

Full Outer Join: (R ⟗ S)

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Courses ⟗ HoD

A	B	C	D
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya
---	---	104	Mira

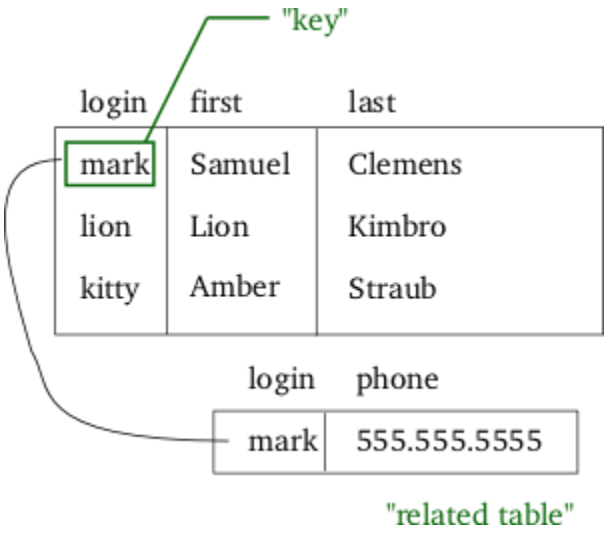
Relational Model

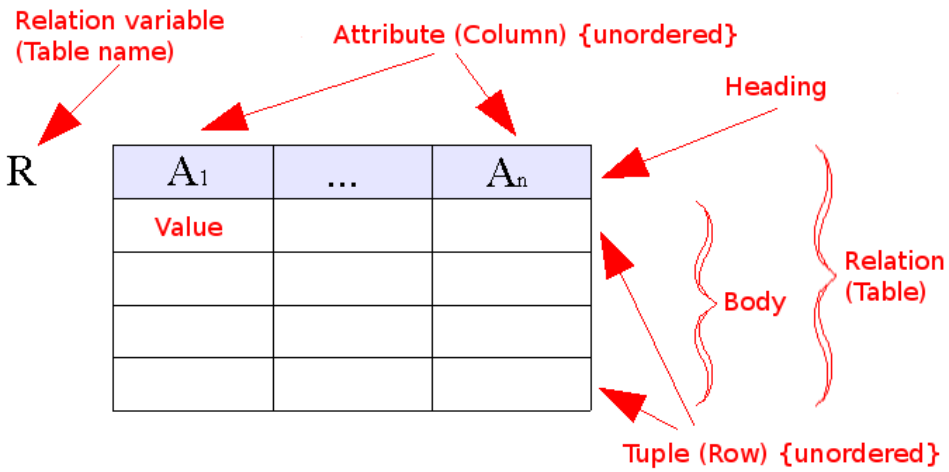
Activity Code	Activity Name
23	Patching
24	Overlay
25	Crack Sealing

Key = 24

Activity Code	Date	Route No.
24	01/12/01	I-95
24	02/08/01	I-66

Date	Activity Code	Route No.
01/12/01	24	I-95
01/15/01	23	I-495
02/08/01	24	I-66





Natural join (⋈)

Employee

Name	EmplId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Employee ⋈ *Dept*

Name	EmplId	DeptName	Manager
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

θ-join and equijoin

Car	
CarModel	**CarPrice**
CarA	20,000
CarB	30,000
CarC	50,000

Boat	
BoatModel	**BoatPrice**
Boat1	10,000
Boat2	40,000
Boat3	60,000

$$Car \bowtie Boat$$
$$CarPrice \geq BoatPrice$$

CarModel	CarPrice	BoatModel	BoatPrice
CarA	20,000	Boat1	10,000
CarB	30,000	Boat1	10,000
CarC	50,000	Boat1	10,000
CarC	50,000	Boat2	40,000

Semijoin (⋉)(⋊)

Employee		
Name	**EmpId**	**DeptName**
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Production

Dept	
DeptName	**Manager**
Sales	Bob
Sales	Thomas
Production	Katie
Production	Mark

Employee ⋉ *Dept*		
Name	**EmpId**	**DeptName**
Sally	2241	Sales
Harriet	2202	Production

Antijoin (▷)

Employee		
Name	**EmpId**	**DeptName**
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Production

Dept	
DeptName	**Manager**
Sales	Sally
Production	Harriet

Employee ▷ Dept		
Name	**EmpId**	**DeptName**
Harry	3415	Finance
George	3401	Finance

Division (÷)

Completed	
Student	**Task**
Fred	Database1
Fred	Database2
Fred	Compiler1
Eugene	Database1
Eugene	Compiler1
Sarah	Database1
Sarah	Database2

DBProject
Task
Database1
Database2

Completed ÷ DBProject
Student
Fred
Sarah

Left outer join (⟕)

Employee		
Name	**EmplId**	**DeptName**
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept	
DeptName	**Manager**
Sales	Harriet
Production	Charles

Employee ⟕ Dept			
Name	**EmplId**	**DeptName**	**Manager**
Harry	3415	Finance	ω
Sally	2241	Sales	Harriet
George	3401	Finance	ω
Harriet	2202	Sales	Harriet
Tim	1123	Executive	ω

Right outer join (⟖)

Employee		
Name	**EmplId**	**DeptName**
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept	
DeptName	**Manager**
Sales	Harriet
Production	Charles

Employee ⟖ Dept			
Name	**EmplId**	**DeptName**	**Manager**
Harry	3415	Finance	ω
Sally	2241	Sales	Harriet
George	3401	Finance	ω
Harriet	2202	Sales	Harriet
Tim	1123	Executive	ω

Full outer join (⟗)

Employee		
Name	**EmplId**	**DeptName**
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept	
DeptName	**Manager**
Sales	Harriet
Production	Charles

Employee ⟗ Dept			
Name	**EmplId**	**DeptName**	**Manager**
Sally	2241	Sales	Harriet
Harriet	2202	Sales	Harriet
ω	ω	Production	Charles

SQL :-

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database. In 1974 the Structured Query Language appeared.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Why SQL

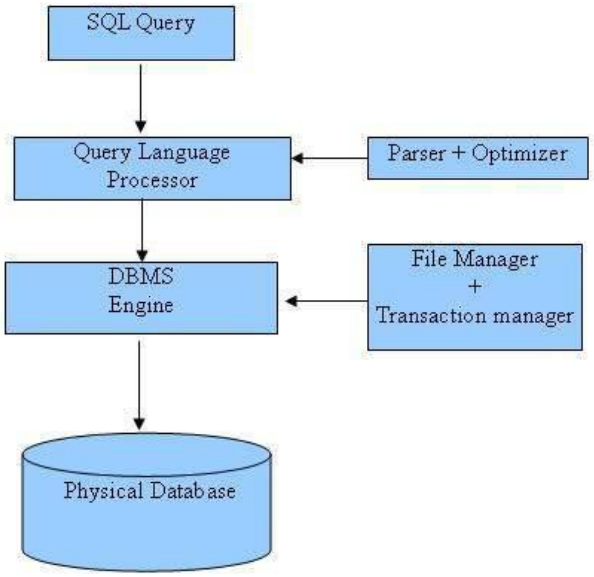
- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures, and views

SQL Process:

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

Following is a simple diagram showing SQL Architecture:



SQL Components

SQL consists of three components:

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Data Control Language (DCL)

The Data Definition Language (DDL):-

It is used to create and modify tables and other objects in the database. For tables there are three main commands:

CREATE TABLE tablename to create a table in the database
DROP TABLE tablename to remove a table from the database
ALTER TABLE tablename to add or remove columns from a table in the database

The Data Manipulation Language (DML):-

It is used to manipulate data within a table. There are four main commands:

SELECT to select rows of data from a table
INSERT to insert rows of data into a table

UPDATE to change rows of data in a table
DELETE to remove rows of data from a table

The Data Control Language (DCL):-

It is used to create privileges to allow users access to, and manipulation of, the database. There are two main commands:

GRANT to grant a privilege to a user
REVOKE to revoke (remove) a privilege from a user

Query optimization

it is a function of many relational database management systems. The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans.

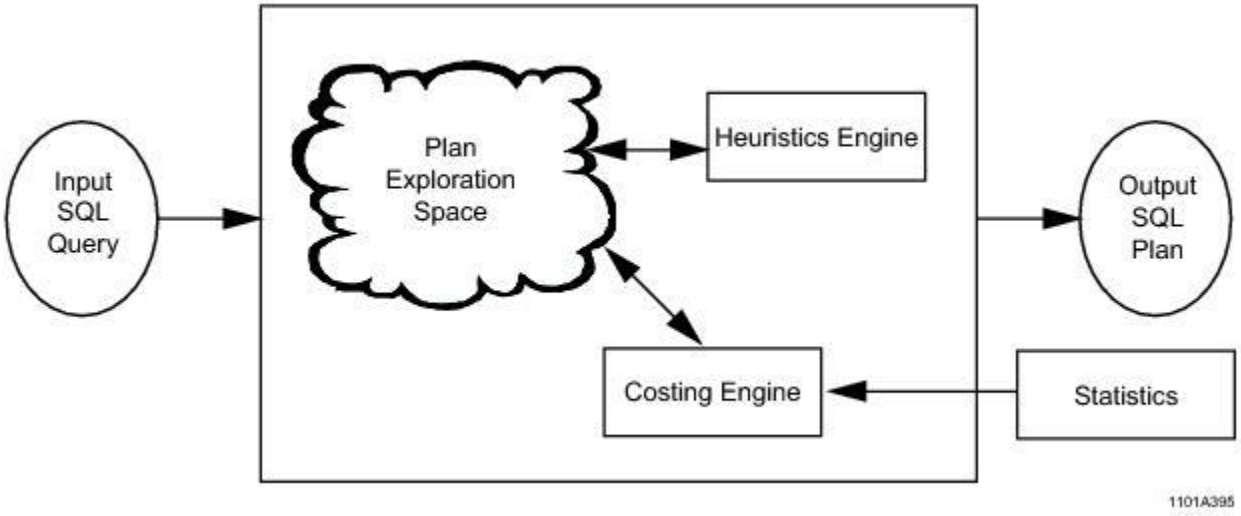
Generally, the query optimizer cannot be accessed directly by users: once queries are submitted to database server, and parsed by the parser, they are then passed to the query optimizer where optimization occurs. However, some database engines allow guiding the query optimizer with hints.

A query is a request for information from a database. It can be as simple as "finding the address of a person with SS# 123-45-6789," or more complex like "finding the average salary of all the employed married men in California between the ages 30 to 39, that earn less than their wives." Queries results are generated by accessing relevant database data and manipulating it in a way that yields the requested information. Since database structures are complex, in most cases, and especially for not-very-simple queries, the needed data for a query can be collected from a database by accessing it in different ways, through different data-structures, and in different orders. Each different way typically requires different processing time. Processing times of the same query may have large variance, from a fraction of a second to hours, depending on the way selected.

The purpose of query optimization which is an automated process, is to find the way to process a given query in minimum time. The large possible variance in time justifies performing query optimization, though finding the exact optimal way to execute a query, among all possibilities, is typically very complex, time consuming by itself, may be too costly, and often practically impossible. Thus query optimization typically tries to approximate the optimum by comparing several common-sense alternatives to provide in a reasonable time a "good enough" plan which typically does not deviate much from the best possible result.

Definition of a Query Optimizer

A query optimizer is an intricate software system that performs several transformations on SQL queries. The following graphic is a very high-level representation of an SQL query optimizer.



The input to the Optimizer, labelled as *Input SQL Query* in the graphic, is actually the ResTree´ (sometimes called ResTree´ or *Red Tree´* to differentiate it from the ResTree/Red Tree as it is configured when it is input to Query Rewrite) output from Query Rewrite, so by this point the original SQL query text has already been reformulated as an annotated parse tree.

The Plan Exploration Space is a workspace where various query and join plans and subplans are generated and costed.

The *Costing* Engine compares the costs of the various plans and subplans generated in the Plan Exploration Space and returns the least costly plan from a given evaluation set .

The Costing Engine primarily uses cardinality estimates based on summary demographic information, whether derived from collected statistics or from dynamic AMP sample estimates, on the set of relations being evaluated to produce its choice of a given “best” plan. The statistical data used to make these evaluations is contained within a set of synoptic data structures called histograms that are maintained in the dictionary.

The Heuristics Engine is a set of rules and guidelines used to evaluate plans and subplans in those situations where cost alone cannot determine the best plan.

Given this framework from which to work, Query Rewrite and the Optimizer perform the following actions in, roughly, the following order.

1. Translate an SQL query into some internal representation.
2. Rewrite the translation into a canonical form. The conversion to canonical form is often referred to as *Query Rewrite*. More accurately, Query Rewrite is a processing step that *precedes* the query processing steps that constitute the canonical process of query optimization.
3. Assess and evaluate candidate procedures for accessing, joining, and aggregating database tables.

Join plans have 3 additional components.

• Selecting a join method.

There are often several possible methods that can be used to make the same join. For example, it is usually, but not always, less expensive to use a Merge Join rather than a Product Join. The choice of a method often has a major effect on the overall cost of processing a query.

• Determining an optimal join geography.

Different methods of relocating rows to be joined can have very different costs. For example, depending on the size of the tables in a join operation, it might be less costly to duplicate one of the tables rather than redistributing it.

• Determining an optimal join order.

Only two tables can be joined at a time. The sequence in which table pairs are joined can have a powerful impact on join cost. In this context, a table could be joined with a spool file rather than another table. The term *table* is used in the most generic sense of the word. Both tables and spool files are frequently categorized using the logical term *relations* when discussing query optimization.

4. Generate several candidate query plans and select the least costly plan from the generated set for execution. A query plan is a set of low-level retrieval instructions called AMP steps that are generated by the Optimizer to produce a query result in the least expensive .

SQL query optimizers are based on principles derived from compiler optimization and from artificial intelligence.

Query tree :-

It is an internal representation of an SQL statement where the single parts that built it are stored separately. These query trees are visible when starting the PostgreSQL backend with debug level 4 and typing queries into the interactive backend interface. The rule

actions in the pg_rewrite system catalog are also stored as query trees. They are not formatted like the debug output, but they contain exactly the same information.

Reading a query tree requires some experience and it was a hard time when I started to work on the rule system. I can remember that I was standing at the coffee machine and I saw the cup in a target list, water and coffee powder in a range table and all the buttons in a qualification expression. Since SQL representations of query trees are sufficient to understand the rule system, this document will not teach how to read them .

canonical form

a canonical, normal, or standard form of a mathematical object is a standard way of presenting that object as a mathematical expression. The distinction between "canonical" and "normal" forms varies by subfield. In most fields, a canonical form specifies a *unique* representation for every object, while a normal form simply specifies its form, without the requirement of uniqueness.

The canonical form of a positive integer in decimal representation is a finite sequence of digits that does not begin with zero.

More generally, for a class of objects on which an equivalence relation is defined, a canonical form consists in the choice of a specific object in each class. For example, Jordan normal form is a canonical form for matrix similarity, and the row echelon form is a canonical form, when one consider as equivalent a matrix and its left product by an invertible matrix.

In computer science, and more specifically in computer algebra, when representing mathematical objects in a computer, there are usually many different ways to represent the same object. In this context, a canonical form is a representation such that every object has a unique representation. Thus, the equality of two objects can easily be tested by testing the equality of their canonical forms. However canonical forms frequently depend on arbitrary choices (like ordering the variables), and this introduces difficulties for testing the equality of two objects resulting on independent computations. Therefore, in computer algebra, *normal form* is a weaker notion: A normal form is a representation such that zero is uniquely represented. This allows testing for equality by putting the difference of two objects in normal form.

Canonical form can also mean a differential form that is defined in a natural (canonical) way.

In computer science, data that has more than one possible representation can often be canonicalized into a completely unique representation called its canonical form. Putting something into canonical form is canonicalization.

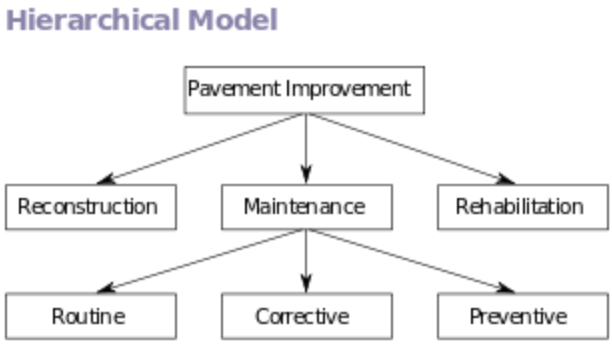
Database model :-

A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated. The most popular example of a database model is the relational model (or the SQL approximation of relational), which uses a table-based format.

Common logical data models for databases include:

1. **Hierarchical database model :-**

A **hierarchical database model** is a data model in which the data is organized into a tree-like structure. The data is stored as **records** which are connected to one another through **links**. A record is a collection of fields, with each field containing only one value. The **entity type** of a record defines which fields the record contains.



Example of a hierarchical model

A record in the hierarchical database model corresponds to a row (or tuple) in the relational database model and an entity type corresponds to a table (or relation).

The hierarchical database model mandates that each child record has only one parent, whereas each parent record can have one or more child records. In order to retrieve data from a hierarchical database the whole tree needs to be traversed starting from the root node. This model is recognized as the first database model created by IBM in the 1960s

Examples of hierarchical data represented as relational tables

An organization could store employee information in a table that contains attributes/columns such as employee number, first name, last name, and department number. The organization provides each employee with computer hardware as needed, but computer equipment may only be used by the employee to which it is assigned. The organization could store the computer hardware information in a separate table that includes each part's serial number, type, and the employee that uses it. The tables might look like this:

employee table				computer table		
EmpNo	**First Name**	**Last Name**	**Dept. Num**	**Serial Num**	**Type**	**User EmpNo**
100	Ram	Prathap	10-L	3009734-4	Computer	100
101	Ravi	Kumar	10-L	3-23-283742	Monitor	100
102	Ramesh	Naidu	20-B	2-22-723423	Monitor	100
103	sarath	yadav	20-B	232342	Printer	100

In this model, the employee data table represents the "parent" part of the hierarchy, while the computer table represents the "child" part of the hierarchy. In contrast to tree structures usually found in computer software algorithms, in this model the children point to the parents. As shown, each employee may possess several pieces of computer equipment, but each individual piece of computer equipment may have only one employee owner.

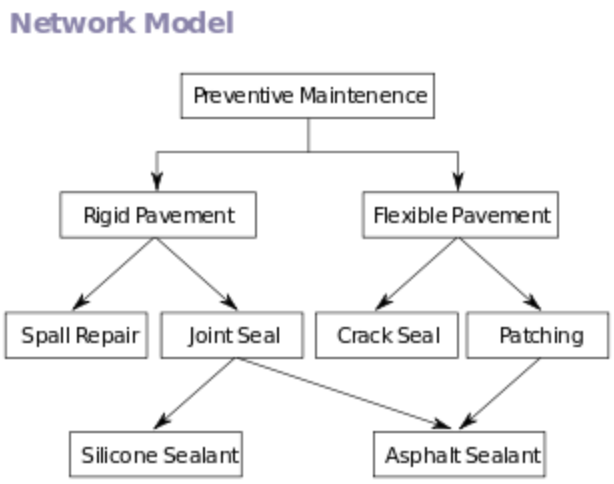
Consider the following structure:

EmpNo	**Designation**	**ReportsTo**
10	Director	
20	Senior Manager	10
30	Typist	20
40	Programmer	20

In this, the "child" is the same type as the "parent". The hierarchy stating EmpNo 10 is boss of 20, and 30 and 40 each report to 20 is represented by the "ReportsTo" column. In Relational database terms, the ReportsTo column is a foreign key referencing the EmpNo column. If the "child" data type were different, it would be in a different table, but there would still be a foreign key referencing the EmpNo column of the employees table.

This simple model is commonly known as the adjacency list model, and was introduced by Dr. Edgar F. Codd after initial criticisms surfaced that the relational model could not model hierarchical data.

2. The **network model** is a database model conceived as a flexible way of representing objects and their relationships. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, is not restricted to being a hierarchy or lattice.



While the hierarchical database model structures data as a tree of records, with each record having one parent record and many children, the network model allows each record to have multiple parent and child records, forming a generalized graph structure. This property applies at two levels: the schema is a generalized graph of record types connected by relationship types (called "set types" in CODASYL), and the database itself is a generalized graph of record occurrences connected by relationships (CODASYL "sets"). Cycles are permitted at both levels. The chief argument in favour of the network model, in comparison to the hierarchic model, was that it allowed a more natural modeling of relationships between entities. Although the model was widely implemented and used, it failed to become dominant for two main reasons. Firstly, IBM chose to stick to the hierarchical model with semi-network extensions in their established products such as IMS and DL/I. Secondly, it was eventually displaced by the relational model, which offered a higher-level, more declarative interface. Until the early 1980s the performance benefits of the low-level navigational interfaces offered by hierarchical and network databases were persuasive for many large-scale applications, but as hardware became faster, the extra productivity and flexibility of the relational model led to the gradual obsolescence of the network model in corporate enterprise usage.