DBMS

Department of Computer Science & Engineering

B. Tech (CSE)

UNIT-1

1.Overview of Database Management System:

Database:

The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently. It is also used to organize the data in the form of a table, schema, views, and reports, etc.

For example: The college Database organizes the data about the admin, staff, students and faculty etc.

Using the database, you can easily retrieve, insert, and delete the information.

Database Management System:

Database management system is a software which is used to manage the database. For example: **MySQL, Oracle**, etc are a very popular commercial database which is used in different applications.

DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.

It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

DBMS allows users the following tasks:

Data Definition: It is used for creation, modification, and removal of definition that defines the organization of data in the database.

Data Updation: It is used for the insertion, modification, and deletion of the actual data in the database.

Data Retrieval: It is used to retrieve the data from the database which can be used by applications for various purposes.

User Administration: It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

Characteristics of DBMS:

- It uses a digital repository established on a server to store and manage the information.
- It can provide a clear and logical view of the process that manipulates data.
- DBMS contains automatic backup and recovery procedures.
- It contains ACID properties which maintain data in a healthy state in case of failure.
- It can reduce the complex relationship between data.
- It is used to support manipulation and processing of data.
- It is used to provide security of data.
- It can view the database from different viewpoints according to the requirements of the user.

Advantages of DBMS OVER FILE PROCESSING SYSTEM:

Controls database redundancy: It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.

Data sharing: In DBMS, the authorized users of an organization can share the data among multiple users.

Easily Maintenance: It can be easily maintainable due to the centralized nature of the database system.

Reduce time: It reduces development time and maintenance need.

Backup: It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.

multiple user interface: It provides different types of user interfaces like graphical user interfaces, application program interfaces

Disadvantages of DBMS:

Cost of Hardware and Software: It requires a high speed of data processor and large memory size to run DBMS software.

Size: It occupies a large space of disks and large memory to run them efficiently.

Complexity: Database system creates additional complexity and requirements.

Higher impact of failure: Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

2. 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.

1-Tier architecture:

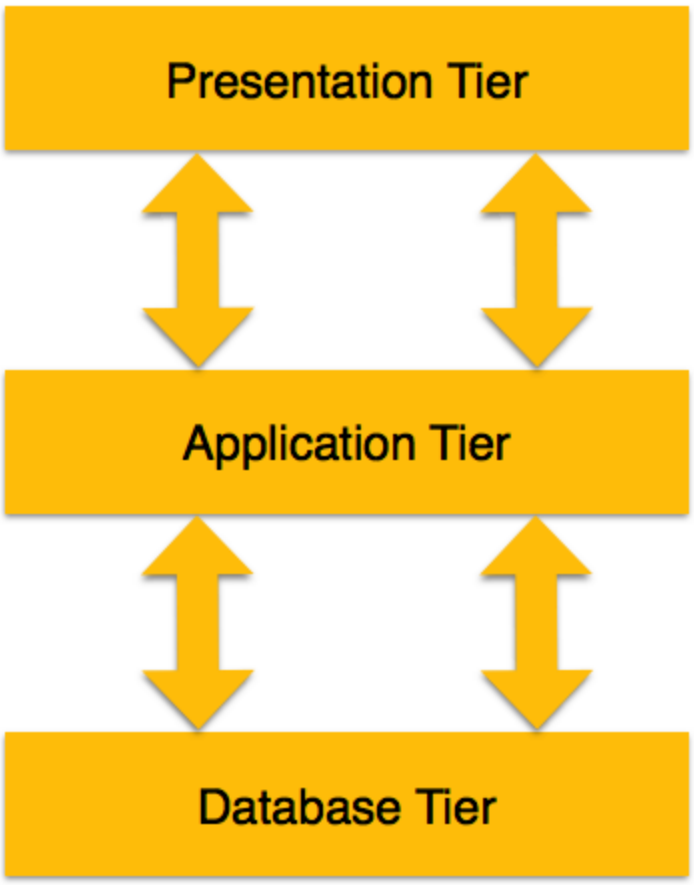
In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-_users. Database designers and programmers normally prefer to use single-tier architecture.

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.

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.



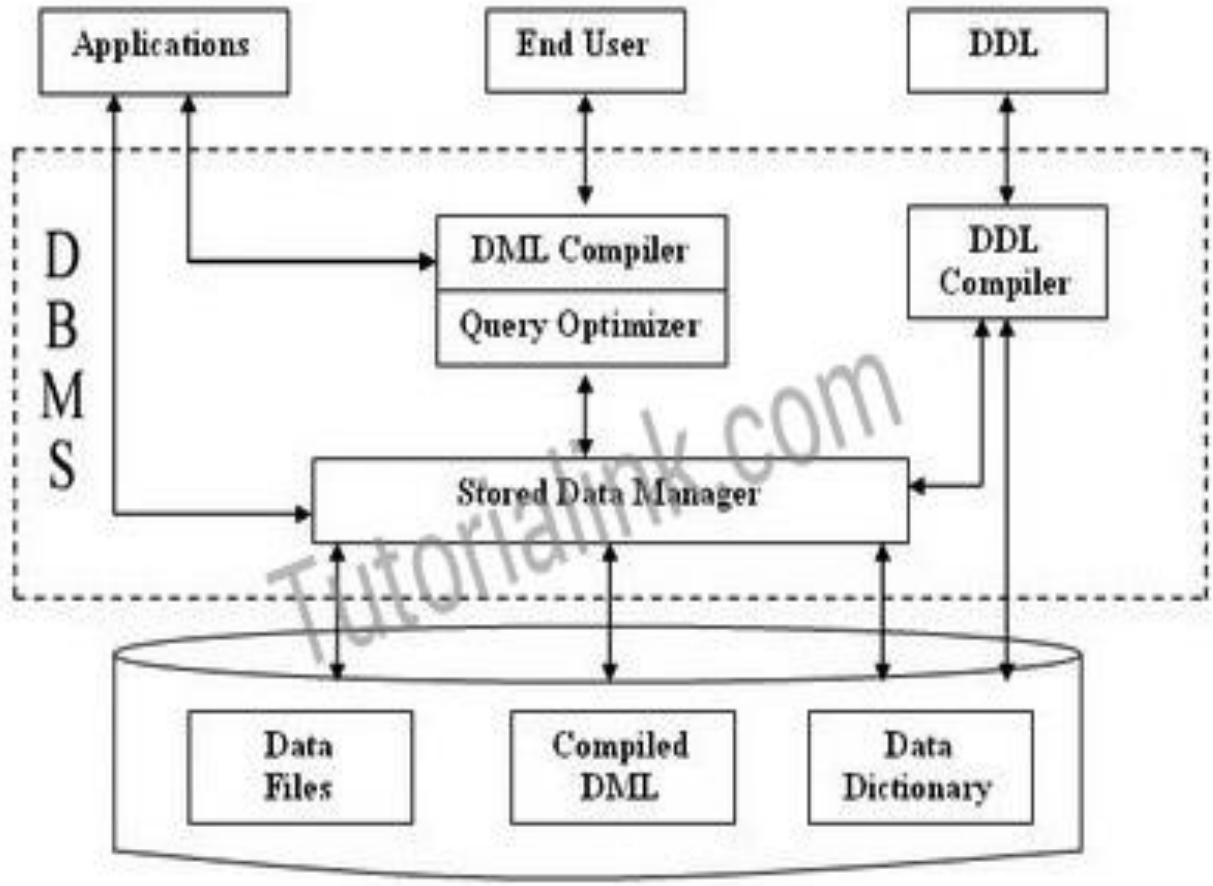
- **Database (Data) Tier** − At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
- **Application (Middle) Tier** − At this tier reside the application server and the programs that access the 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.
- **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.

3. DBMS Component Modules and structure:

Structure of DBMS:

- DBMS (Database Management System) acts as an interface between the user and the database. The user requests the DBMS to perform various operations such as insert, delete, update and retrieval on the database.
- The components of DBMS perform these requested operations on the database and provide necessary data to the users.



- The various components of DBMS are described below:

Components of a DBMS

- The components of DBMS can be divided into two parts:

Function and Services of DBMS:

1. **DDL Compiler:**
    - Data Description Language compiler processes schema definitions specified in the DDL.
    - It includes metadata information such as the name of the files, data items, storage details of each file, mapping information and constraints etc.
2. **DML Compiler and Query optimizer:**
    - The DML commands such as insert, update, delete, retrieve from the application program are sent to the DML compiler for compilation into object code for database access.
    - The object code is then optimized in the best way to execute a query by the query optimizer and then send to the data manager.
3. **Data Manager:**
    - The Data Manager is the central software component of the DBMS also knows as Database Control System.
    - The Main Functions Of Data Manager Are:
        1. Convert operations in user's Queries coming from the application programs or combination of DML Compiler and Query optimizer which is known as Query Processor from user's logical view to physical file system.
        2. Controls DBMS information access that is stored on disk.

3. It also controls handling buffers in main memory.
4. It also enforces constraints to maintain consistency and integrity of the data.
5. It also synchronizes the simultaneous operations performed by the concurrent users.
6. It also controls the backup and recovery operations.

4. **Data Dictionary**:
    - Data Dictionary, which stores metadata about the database, in particular the schema of the database.
    - names of the tables, names of attributes of each table, length of attributes, and number of rows in each table.
    - Detailed information on physical database design such as storage structure, access paths, files and record sizes.
    - Usage statistics such as frequency of query and transactions.
    - Data dictionary is used to actually control the data integrity, database operation and accuracy. It may be used as a important part of the DBMS
5. **Data Files:**
    - Which store the database itself.
6. **Compiled DML:**
    - The DML complier converts the high level Queries into low level file access commands known as compiled DML.
7. **End Users:**
    - The second class of users then is end user, who interacts with system from online workstation or terminals.
    - Use the interface provided as an integral part of the database system software.

3. User can request, in form of query, to access database either directly by using particular language, such as SQL, or by using some pre-developed application interface.
4. Such request are sent to query evaluation engine via DML pre-compiler and DML compiler
5. The query evaluation engine accepts the query and analyses it.
6. It finds the suitable way to execute the compiled SQL statements of the query.
7. Finally, the compiled SQL statements are executed to perform the specified operation

8. Query Processor Units:

Interprets DDL statements into a set of tables containing metadata.
Translates DML statements into low level instructions that the query evaluation engine understands.
Converts DML statements embedded in an application program into procedure calls int he host language.
Executes low level instructions generated by DML compiler.

a. DDL Interpreter
b. DML Compiler
c. Embedded DML Pre-compiler
d. Query Evalution Engine

9. Storage Manager Units

Checks the authority of users to access data.
Checks for the satisfaction of the integrity constraints.
Preserves atomicity and controls concurrency.
Manages allocation of splace on disk.

Fetches data from disk storage to memory for being used.

a.Authorization Manager

b.Integrity Manager

c.Transaction Manager

d.File manager

e.Buffer Manager

4. Three Levels Database Architecture:

Following are the three levels of database architecture,

1. Physical Level
2. Conceptual Level
3. External Level

three levels database architecture

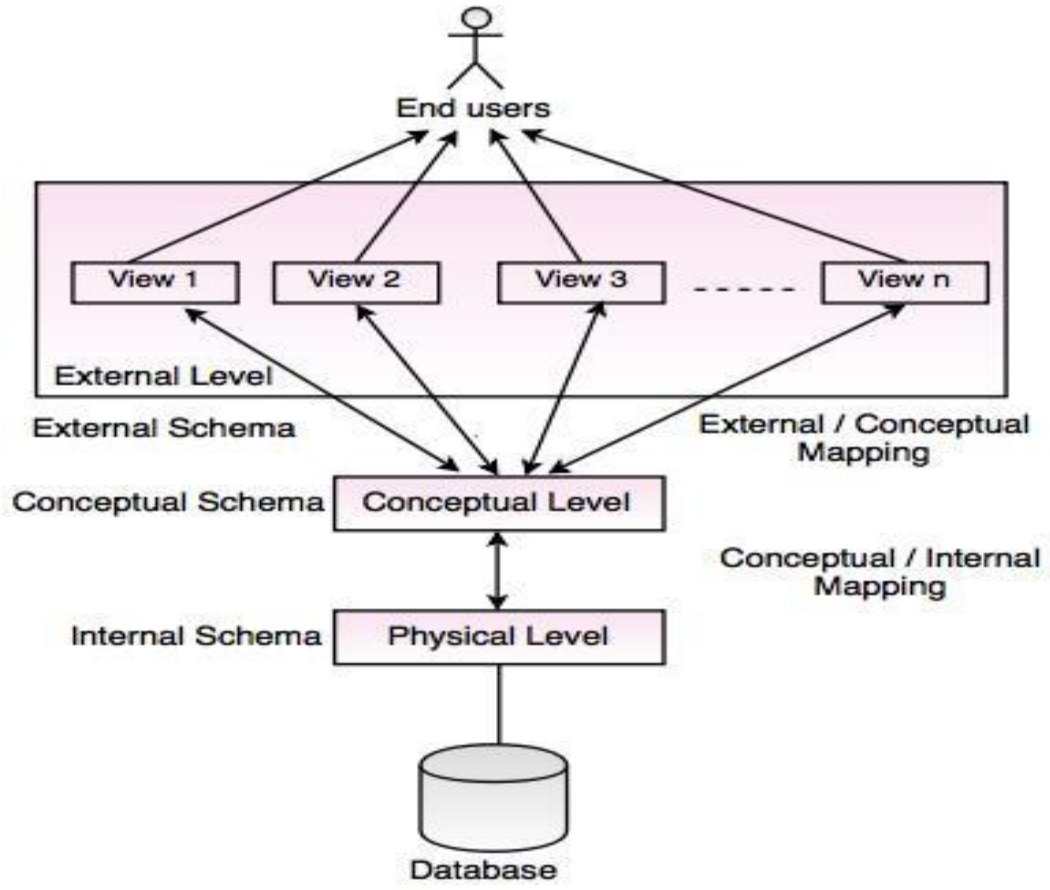


Fig. Three Level Architechture of DBMS

In the above diagram,

It shows the architecture of DBMS.

Mapping is the process of transforming request response between various database levels of architecture.

Mapping is not good for small database, because it takes more time.

In External / Conceptual mapping, DBMS transforms a request on an external schema against the conceptual schema.

In Conceptual / Internal mapping, it is necessary to transform the request from the conceptual to internal levels.

1**. Physical Level:**

Physical level describes the physical storage structure of data in database.
It is also known as Internal Level.
This level is very close to physical storage of data.
At lowest level, it is stored in the form of bits with the physical addresses on the secondary storage device.
At highest level, it can be viewed in the form of files.
The internal schema defines the various stored data types. It uses a physical data model.

2. **Conceptual Level:**

Conceptual level describes the structure of the whole database for a group of users.
It is also called as the data model.
Conceptual schema is a representation of the entire content of the database.
These schema contains all the information to build relevant external records.
It hides the internal details of physical storage.

3. **External Level:**

External level is related to the data which is viewed by individual end users.
This level includes a no. of user views or external schemas.
This level is closest to the user.
External view describes the segment of the database that is required for a particular user group and hides the rest of the database from that user group.

5.Schema and Instance:

- The data which is stored in the database at a particular moment of time is called an instance of the database.
- The overall design of a database is called schema.
- A database schema is the skeleton structure of the database. It represents the logical view of the entire database.
- A schema contains schema objects like table, foreign key, primary key, views, columns, data types, stored procedure, etc.
- A schema diagram can display only some aspects of a schema like the name of record type, data type, and constraints.
- Example:

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

6.Data Independence:

One of the biggest advantages of database is data independence. It means we can change the conceptual schema at one level without affecting the data at other level. It means we can change the structure of a database without affecting the data required by users and program. This feature was not available in file oriented approach. There are two types of data independence and they are:

1. Physical data independence
2. Logical data independence

Data Independence The ability to modify schema definition in on level without affecting schema definition in the next higher level is called data independence. There are two levels of data independence:

1. **Physical data independence** is the ability to modify the physical schema without causing application programs to be rewritten. Modifications at the physical level are occasionally necessary to improve performance. It means we change the physical storage/level without affecting the conceptual or external view of the data. The new changes are absorbed by mapping techniques.

2. **Logical data independence** in the ability to modify the logical schema without causing application program to be rewritten. Modifications at the logical level are necessary whenever the logical structure of the database is altered (for example, when money-market accounts are added to banking system).

Logical Data independence means if we add some new columns or remove some columns from table then the user view and programs should not changes. It is called the logical independence. For example: consider two users A & B. Both are selecting the empno and ename. If user B add a new column salary in his view/table then it will not effect the external view user; user A, but internal view of database has been changed for both users A & B. Now user A can also print the salary.

7.Data Models:

A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system. While the **Relational Model** is the most widely used database model, there are other models too:

- Hierarchical Model
- Network Model
- Entity-relationship Model
- Relational Model

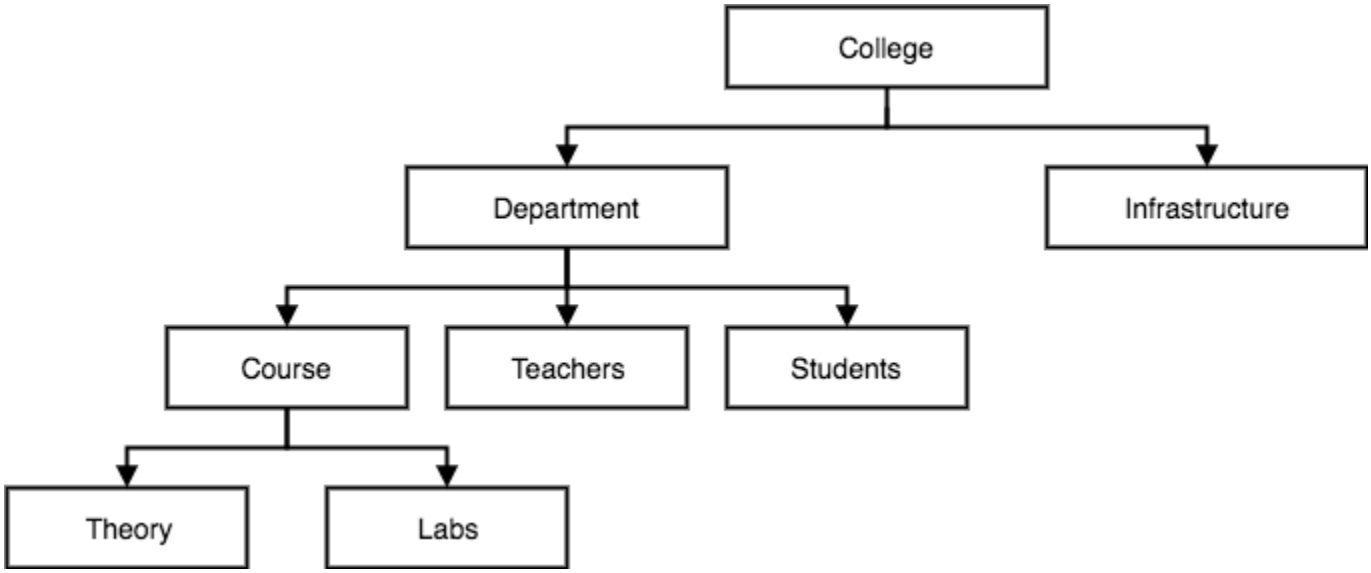
Hierarchical Model

This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. The heirarchy starts from the **Root** data, and expands like a tree, adding child nodes to the parent nodes.

In this model, a child node will only have a single parent node.

This model efficiently describes many real-world relationships like index of a book, recipes etc.

In hierarchical model, data is organised into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.

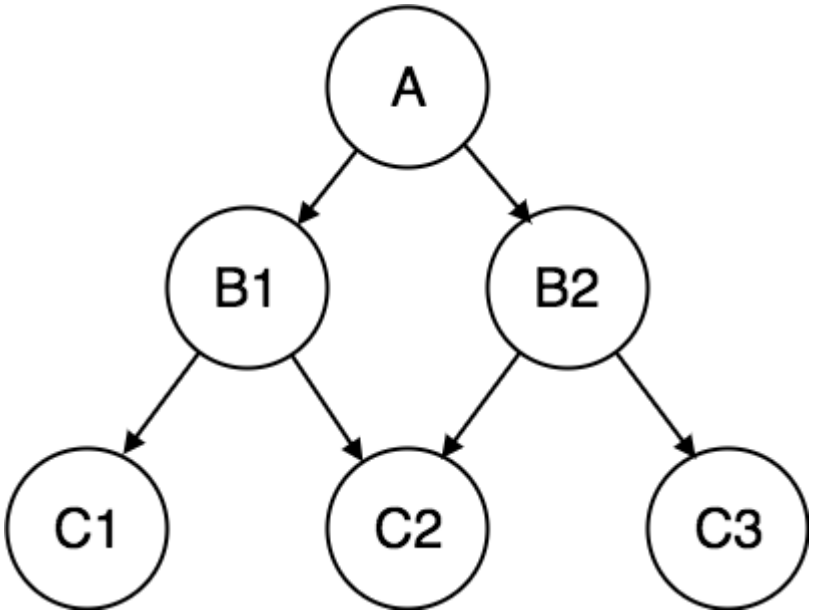


Network Model

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before Relational Model was introduced.



Entity-relationship Model

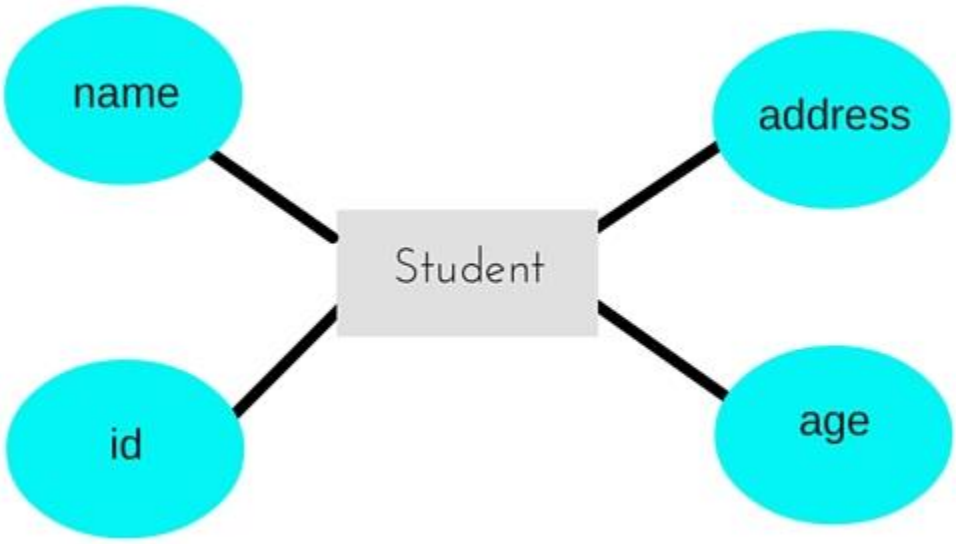
In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.

Different entities are related using relationships.

E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand.

This model is good to design a database, which can then be turned into tables in relational model(explained below).

Let's take an example, If we have to design a School Database,
then **Student** will be an **entity** with **attributes** name, age, address etc.
As **Address** is generally complex, it can be
another **entity** with **attributes** street name, pincode, city etc, and there will be a relationship between them.



Relational Model

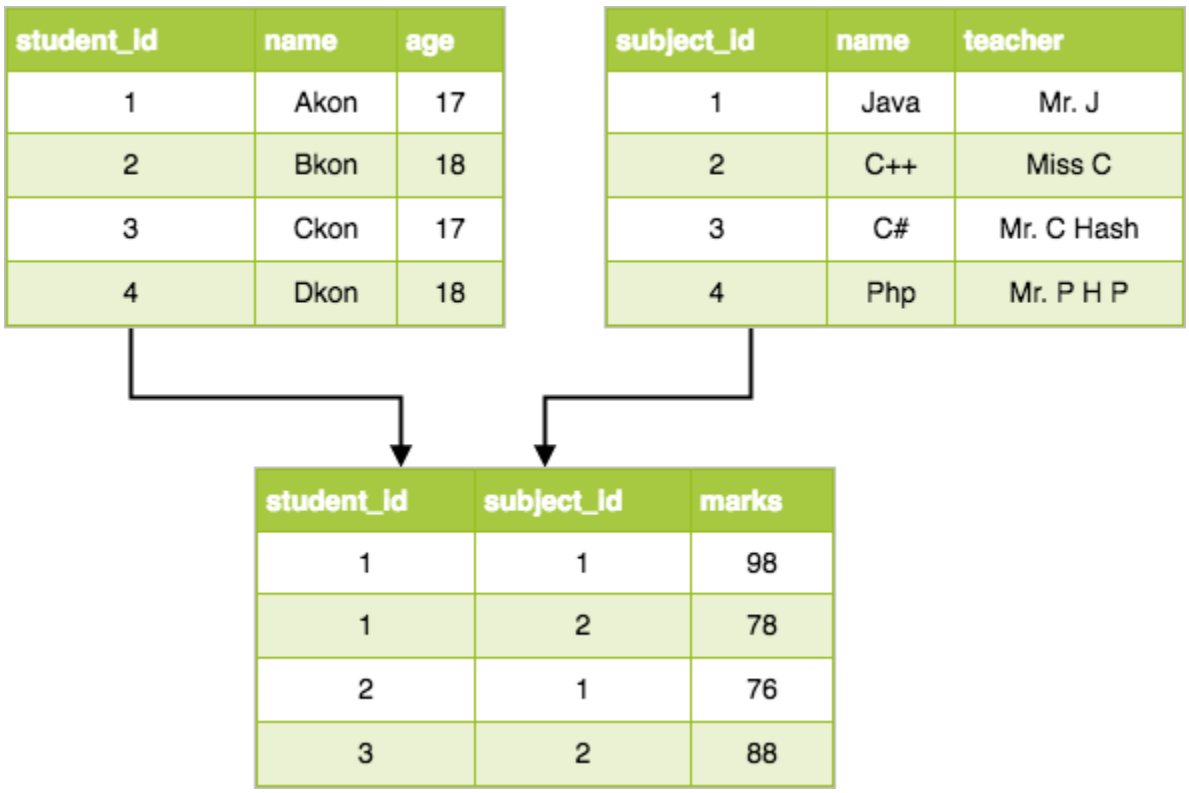
In this model, data is organised in two-dimensional **tables** and the relationship is maintained by storing a common field.

This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model, infact, we can say the only database model used around the world.

The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table.

Hence, tables are also known as **relations** in relational model.

In the coming tutorials we will learn how to design tables, normalize them to reduce data redundancy and how to use Structured Query language to access data from tables.



student_id	name	age
1	Akon	17
2	Bkon	18
3	Ckon	17
4	Dkon	18

subject_id	name	teacher
1	Java	Mr. J
2	C++	Miss C
3	C#	Mr. C Hash
4	Php	Mr. P H P

student_id	subject_id	marks
1	1	98
1	2	78
2	1	76
3	2	88

8. All Database languages:

Data Definition Language (DDL):

statements are used to classify the database structure or schema. It is a type of language that allows the DBA or user to depict and name those entities, attributes, and relationships that are required for the application along with any associated integrity and security constraints. Here are the lists of tasks that come under DDL:

• CREATE - used to create objects in the database.

• ALTER - used to alters the structure of the database.

• DROP - used to delete objects from the database.

• TRUNCATE - used to remove all records from a table, including all spaces allocated for the records are removed.

• COMMENT - used to add comments to the data dictionary • RENAME - used to rename an object.

Data Manipulation Language:

A language that offers a set of operations to support the fundamental data manipulation operations on the data held in the database. Data Manipulation Language (DML) statements are used to manage data within schema objects. Here are the lists of tasks that come under DML:

• SELECT - It retrieves data from a database.

• INSERT - It inserts data into a table.

• UPDATE - It updates existing data within a table.

• DELETE - It deletes all records from a table, the space for the records remain.

• MERGE - UPSERT operation (insert or update).

• CALL - It calls a PL/SQL or Java subprogram.

• EXPLAIN PLAN - It explains access path to data.

• LOCK TABLE - It controls concurrency.

Data Control Language:

There are another two forms of database sub-languages. The Data Control Language (DCL) is used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges. Privileges are of two types,

• System - creating a session, table, etc. are all types of system privilege.

• Object - any command or query to work on tables comes under object privilege. DCL is used to define two commands. These are:

• Grant - It gives user access privileges to a database.

• Revoke - It takes back permissions from the user.

Transaction Control Language (TCL):

Transaction Control statements are used to run the changes made by DML statements. It allows statements to be grouped into logical transactions.

• COMMIT - It saves the work done.

• SAVEPOINT - It identifies a point in a transaction to which you can later roll back.

• ROLLBACK - It restores the database to original since the last COMMIT.

• SET TRANSACTION - It changes the transaction options like isolation level and what rollback segment to use.

9.Data Modeling using the Entity Relationship Model:

Entity-Relationship Model:

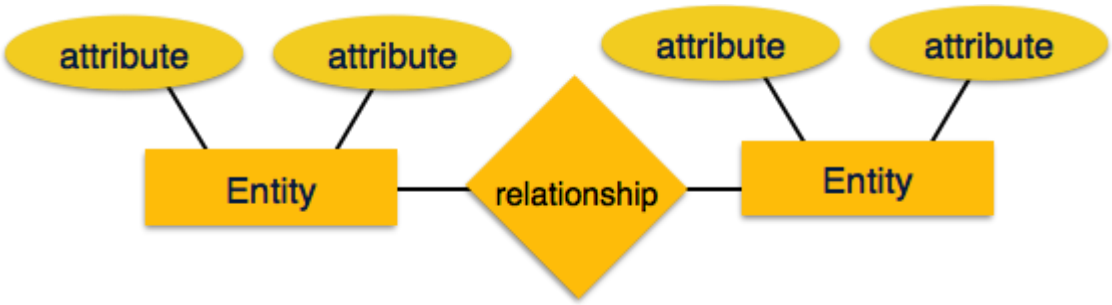
Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

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**.**



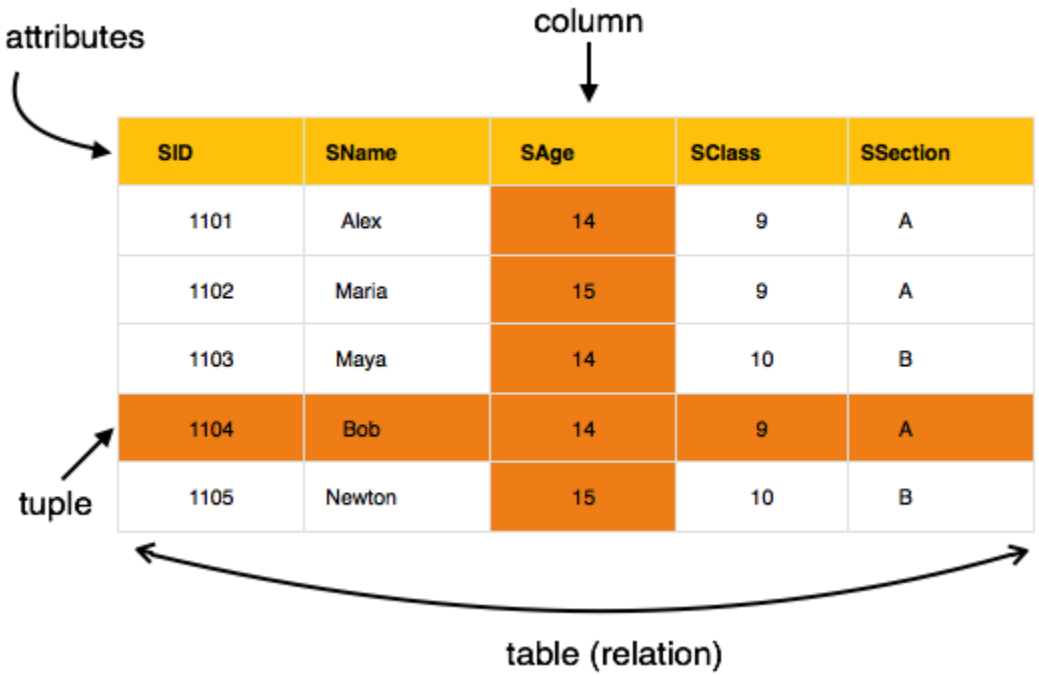
- **Entity** − An entity in an ER Model is a real-world entity having properties called attributes. Every attribute is defined by its set of values called domain. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.
- **Relationship** − The logical association among entities is called *relationship*. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

Mapping cardinalities −

- **one to one**
- **one to many**
- **many to one**
- **many to many**

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.

10.ER model concepts:

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

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. All

these entities have some attributes or properties that give them their identity.

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.

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.

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.
- **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.
- **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.

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.

Relationship:

The association among entities is called a relationship. For example, an employee works_at a department, a student enrolls in a course. Here, Works_at and Enrolls are called relationships.

Relationship Set:

A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

Degree of Relationship:

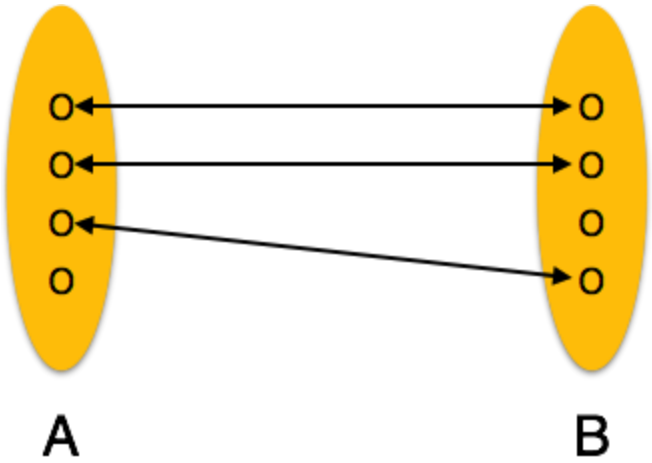
The number of participating entities in a relationship defines the degree of the relationship.

- Binary = degree 2
- Ternary = degree 3
- n-ary = degree

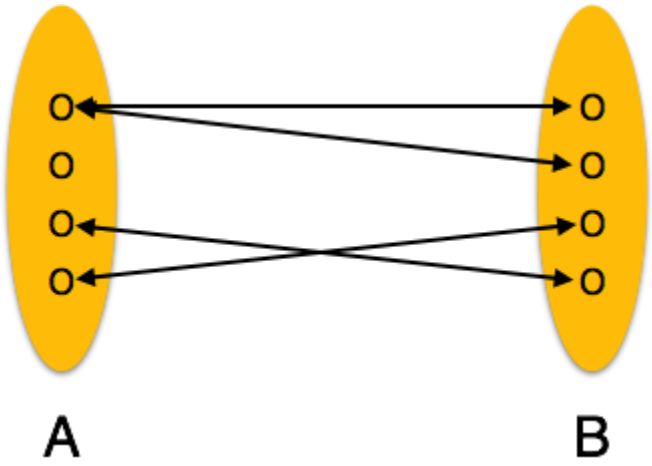
Mapping Cardinalities:

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.

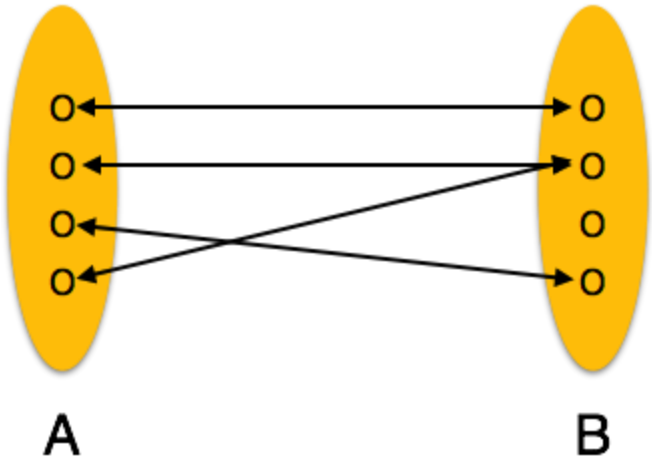
- One-to-one − One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



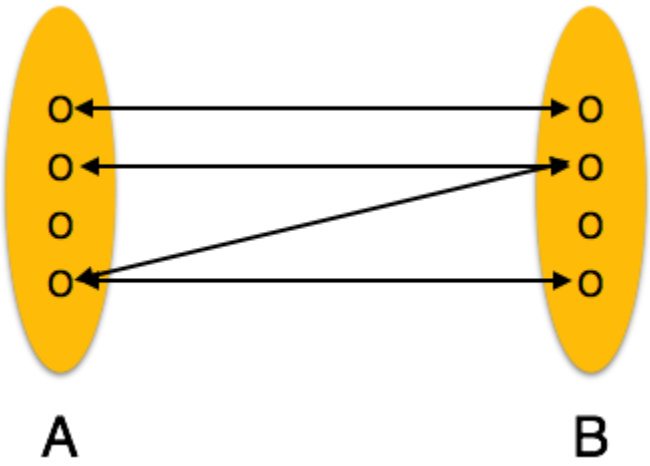
- **One-to-many** − One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



- **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.



11.ER Diagram Representation:

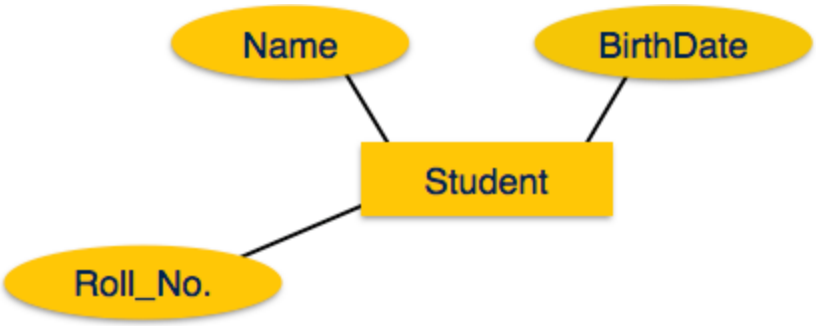
Any object, for example, entities, attributes of an entity, relationship sets, and attributes of relationship sets, can be represented with the help of an ER diagram.

Entity

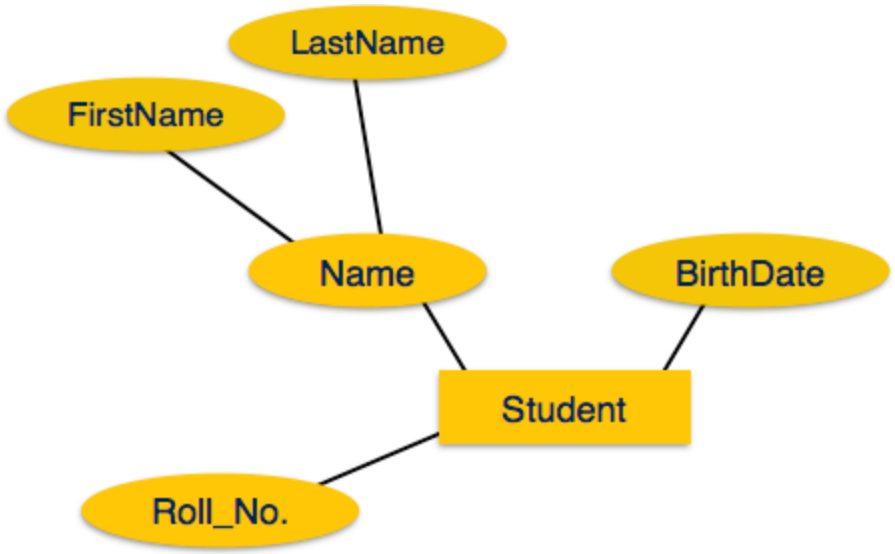
Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

Attributes

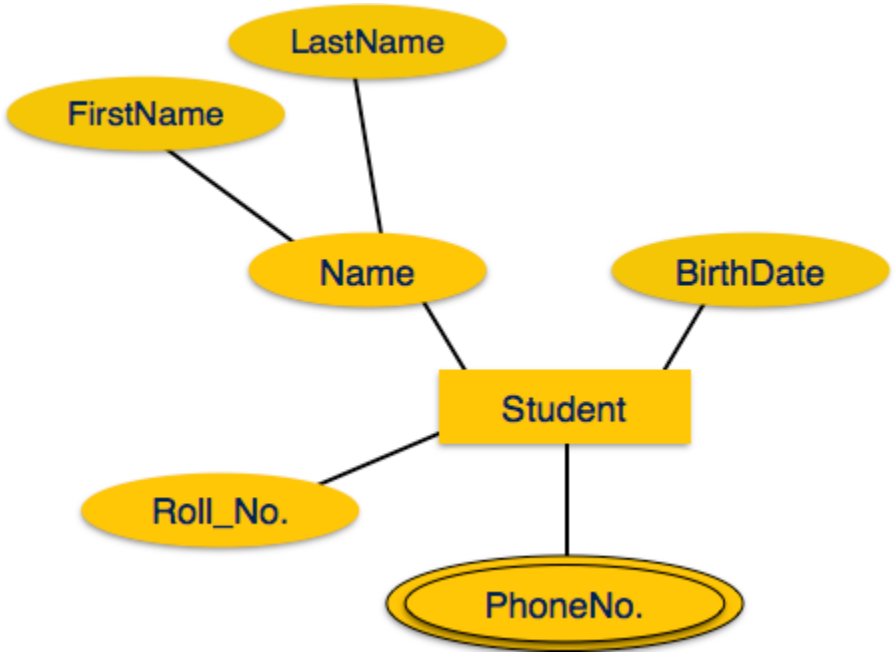
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



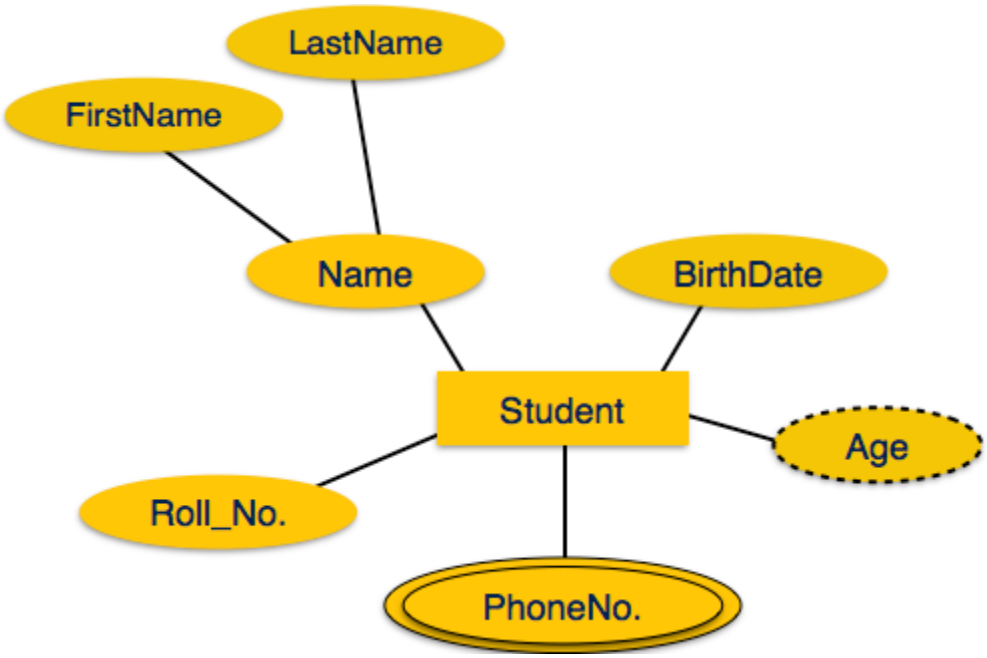
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



Multivalued attributes are depicted by double ellipse.



Derived attributes are depicted by dashed ellipse.



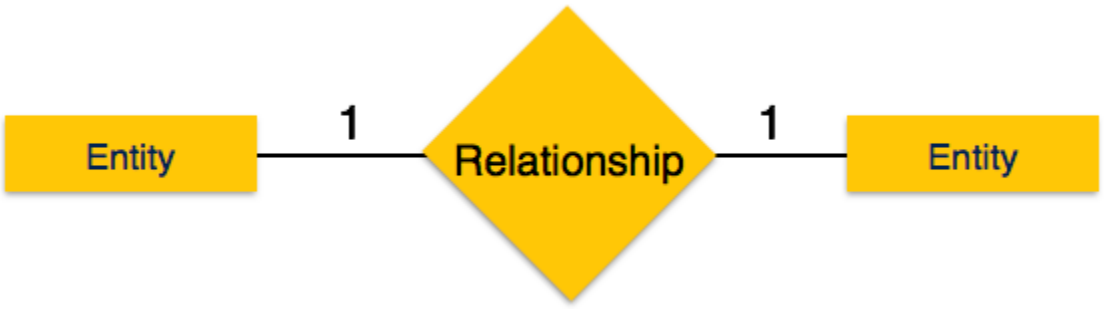
Relationship:

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.

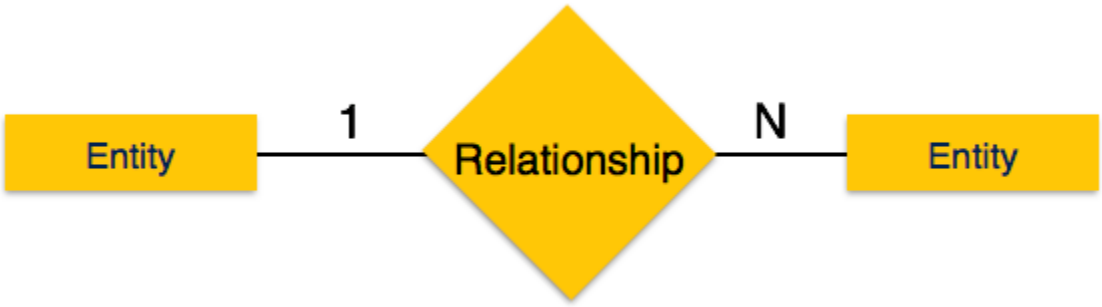
Binary Relationship and Cardinality:

A relationship where two entities are participating is called a **binary relationship**. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

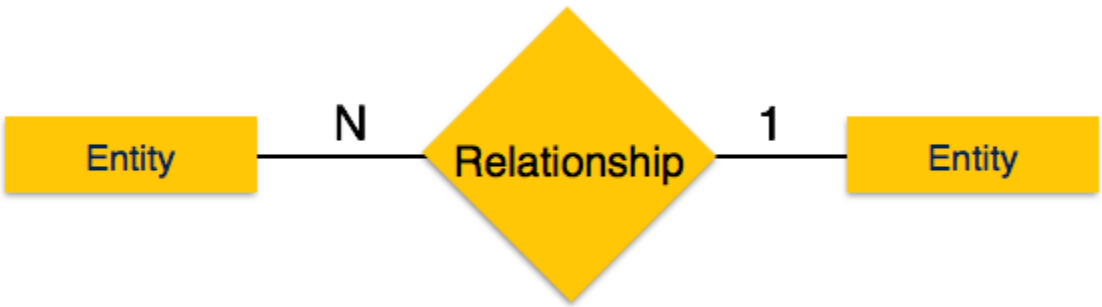
- **One-to-one** − When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



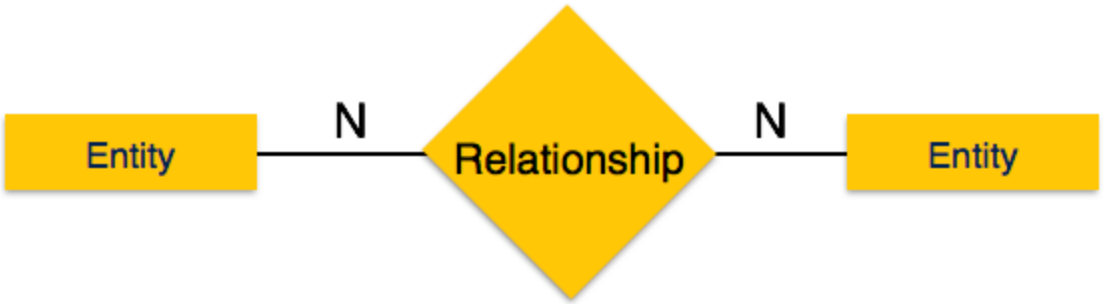
- **One-to-many** − When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.



- **Many-to-one** − When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.

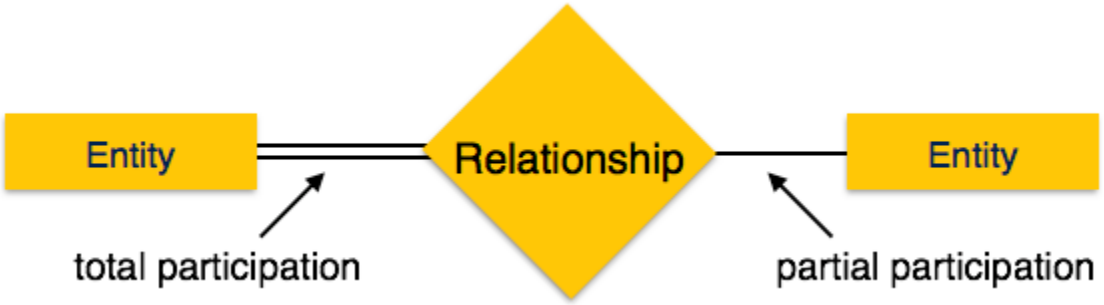


- **Many-to-many** − The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.



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.



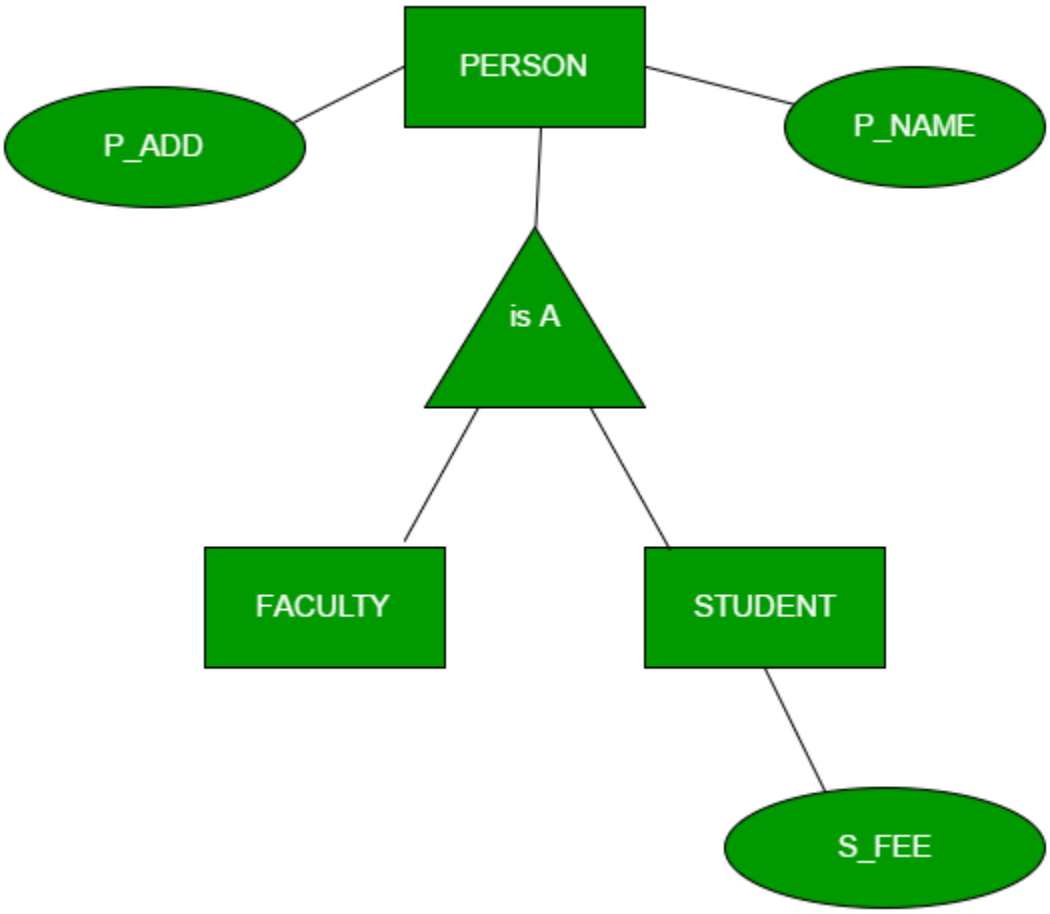
12.Generalization,Specialization & Aggregration:

Generalization, Specialization and Aggregation in ER model are used for data abstraction in which abstraction mechanism is used to hide details of a set of objects.

Generalization –

Generalization is the process of extracting common properties from a set of entities and create a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher level entity if they have some attributes in

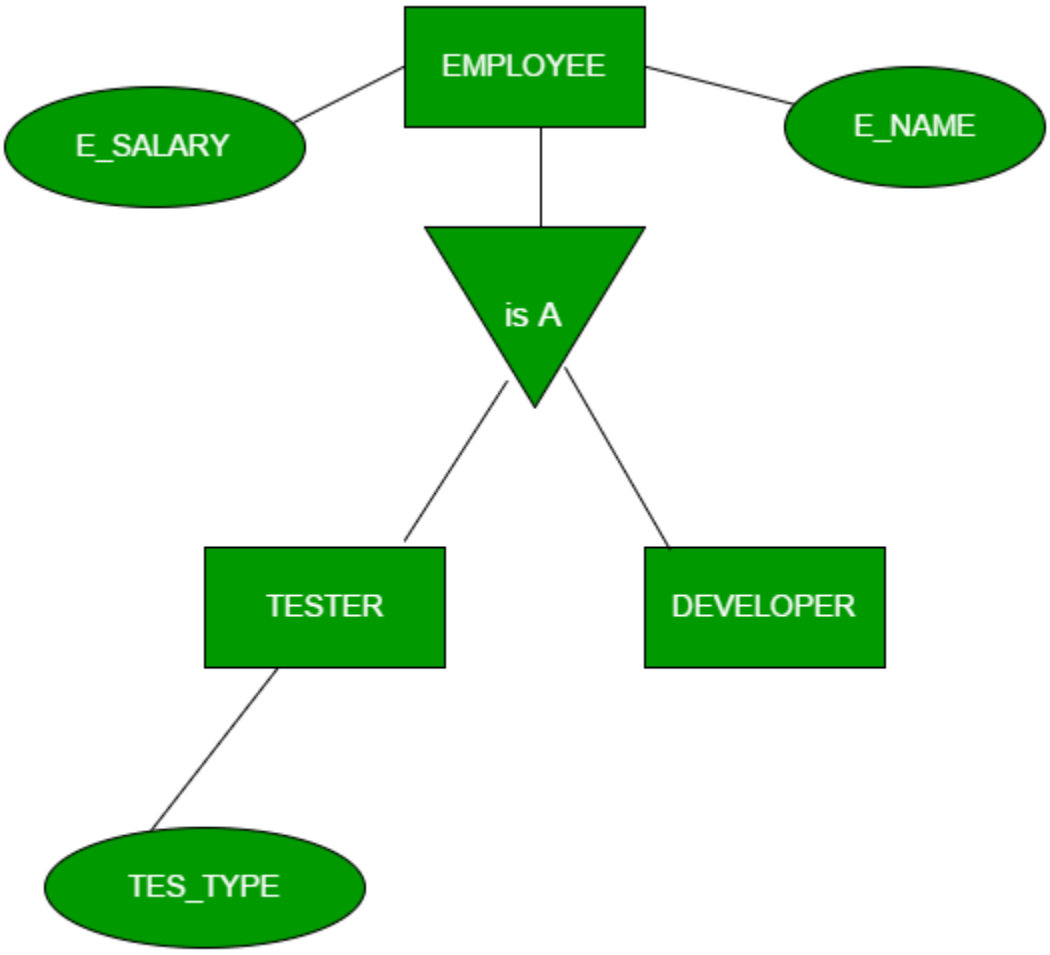
common. For Example, STUDENT and FACULTY can be generalized to a higher level entity called PERSON as shown in Figure . In this case, common attributes like P_NAME, P_ADD become part of higher entity (PERSON) and specialized attributes like S_FEE become part of specialized entity (STUDENT).



Specialization –

In specialization, an entity is divided into sub-entities based on their characteristics. It is a top-down approach where higher level entity is specialized into two or more lower level entities. For Example, EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER etc. as shown in Figure . In this case, common attributes like E_NAME, E_SAL etc.

become part of higher entity (EMPLOYEE) and specialized attributes like TES_TYPE become part of specialized entity (TESTER).

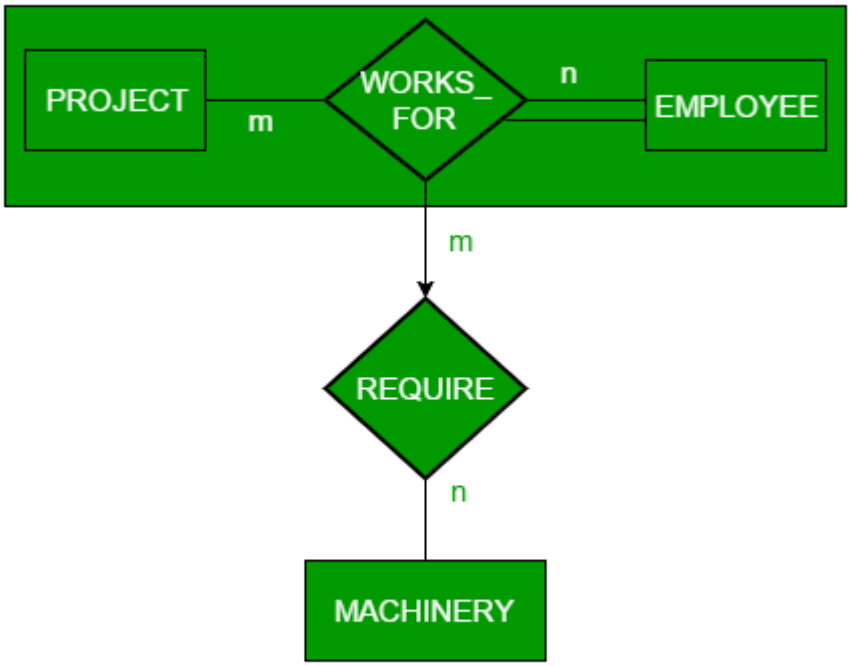


Specialization

Aggregation –

An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher level entity. For Example, Employee working for a project may require some machinery. So, REQUIRE relationship is needed between

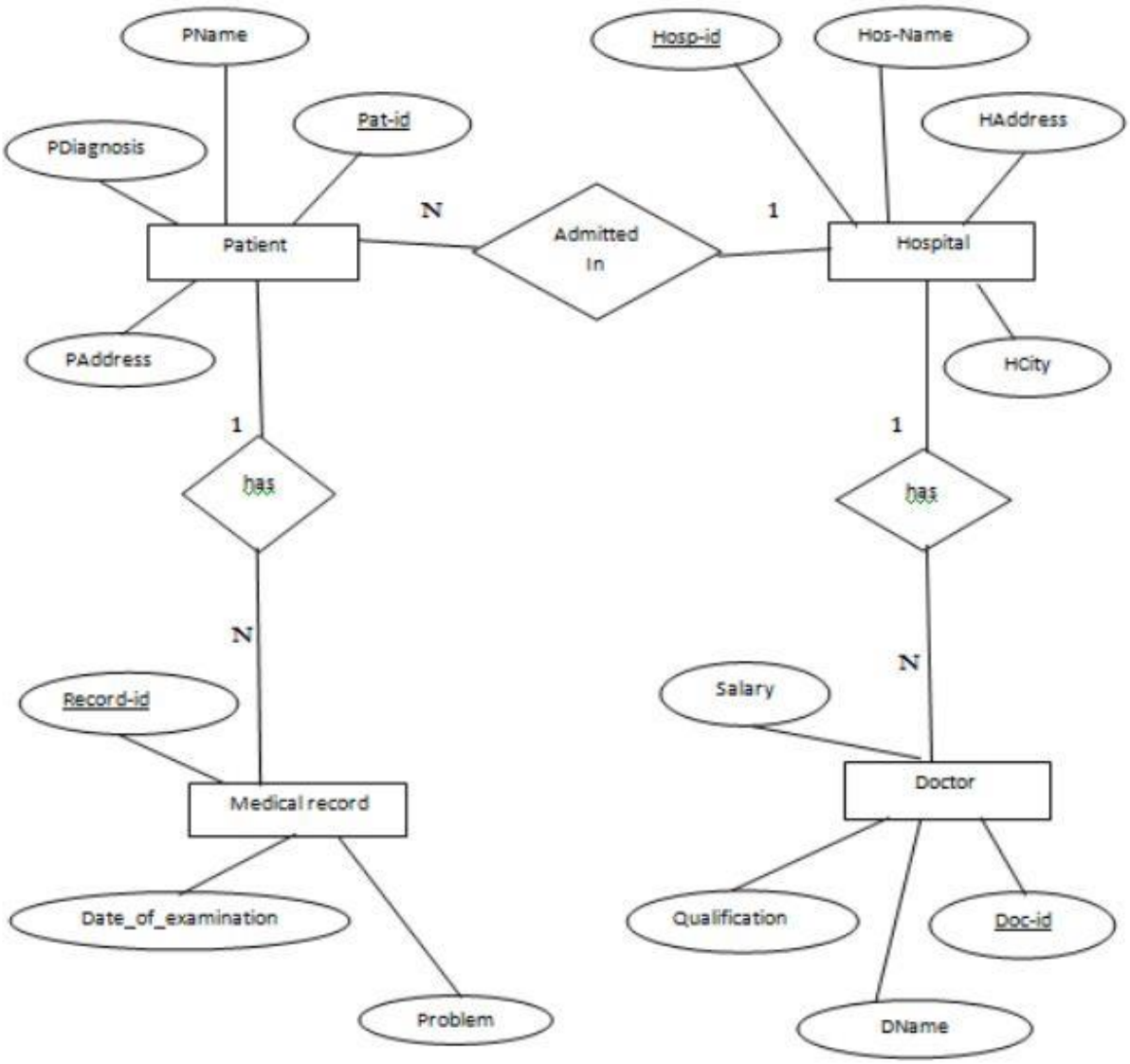
relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into single entity and relationship REQUIRE is created between aggregated entity and MACHINERY.



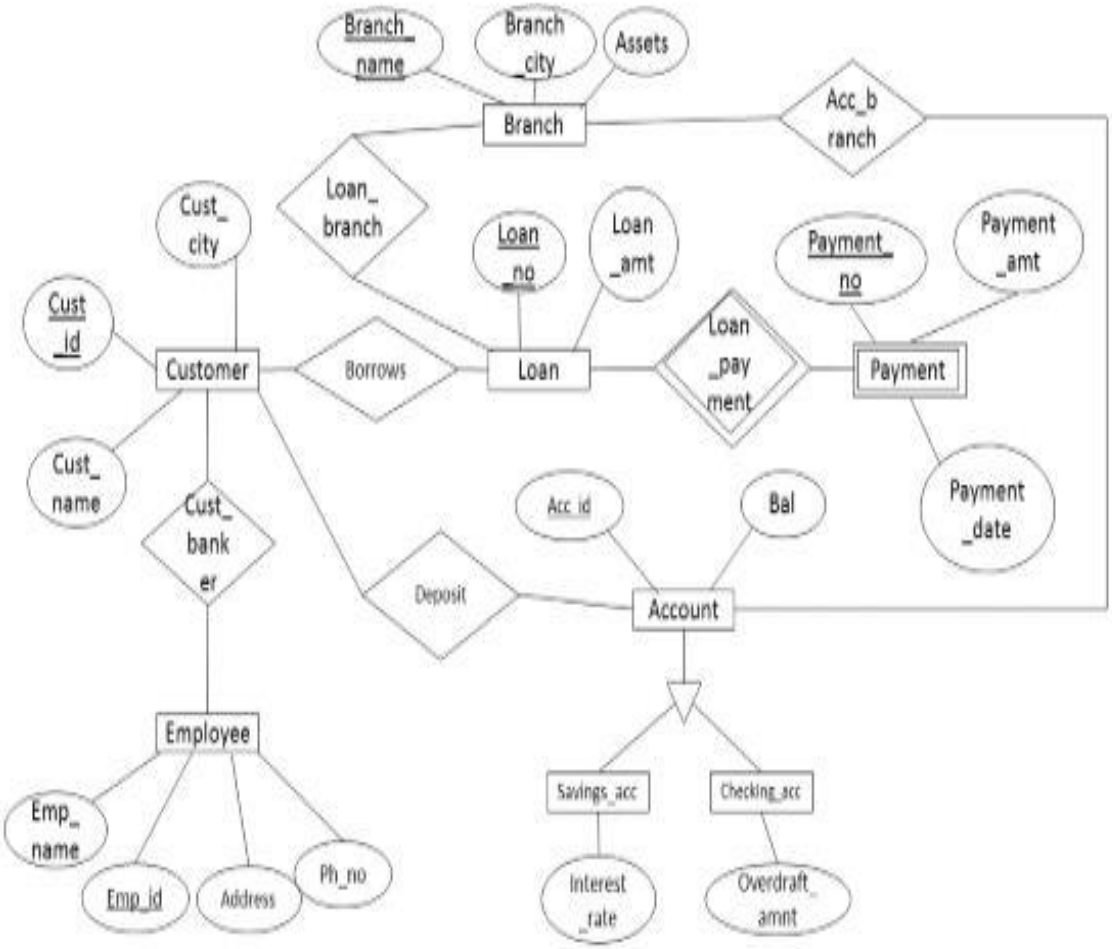
Aggregation

13.Examples of ER Daigram:

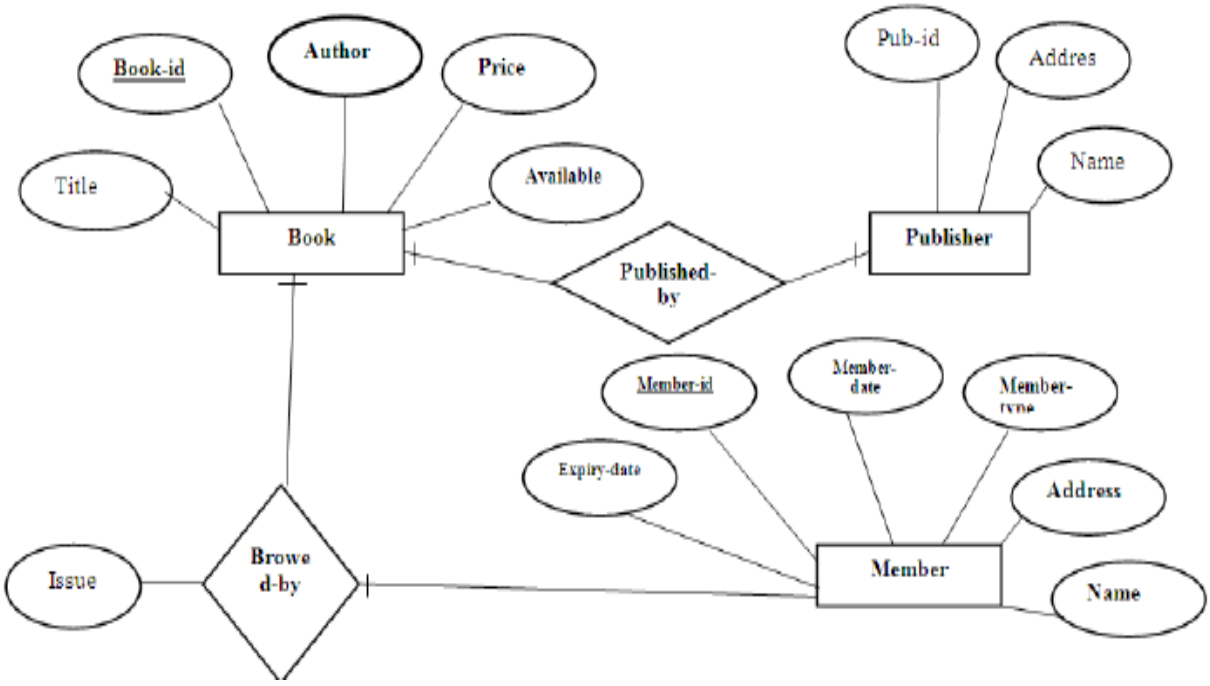
1.Hospital Management System:



2.Bank Management System:



3.Library Management System:


Book-id
Author
Price
Pub-id
Addres
Title
Available
Name
Book
Publisher
Published-by
Member-id
Member-date
Member-type
Expiry-date
Address
Issue
Browe d-by
Member
Name

THANK YOU

DBMS

Department of Computer Science & Engineering

B. Tech (CSE)

UNIT-2

Relational Algebra & Calculus:

1. Preliminaries:

A query language is a language in which user requests to retrieve some information from the database. The query languages are considered as higher level languages than programming languages.

Query languages are of two types, Procedural Language Non-Procedural Language

1. In procedural language, the user has to describe the specific procedure to retrieve the information from the database. Example: The Relational Algebra is a procedural language.

2. In non-procedural language, the user retrieves the information from the database without describing the specific procedure to retrieve it. Example: The Tuple Relational Calculus and the Domain Relational Calculus are non-procedural languages.

EF.Codd's 12 Rules:

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

Relation Data Model:

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.

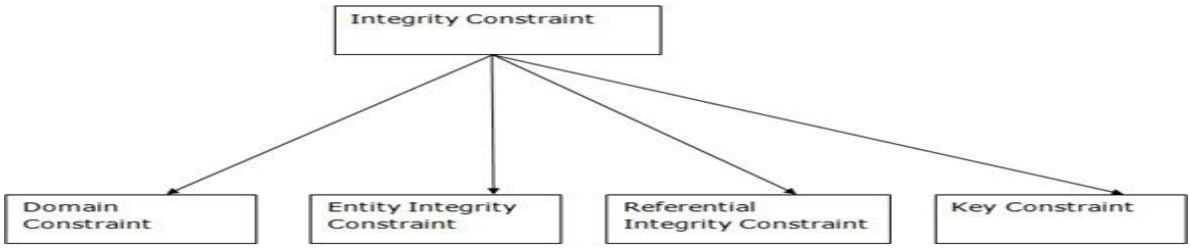
Relation key – Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

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

Types of Integrity Constraint :

- Integrity constraints are set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

Types of Integrity Constraint



1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency etc. The value of attribute must be available in the corresponding domain.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in a relation and if the primary key has null value then we can't identify those rows.
- A table can contain null value other than primary key field.

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

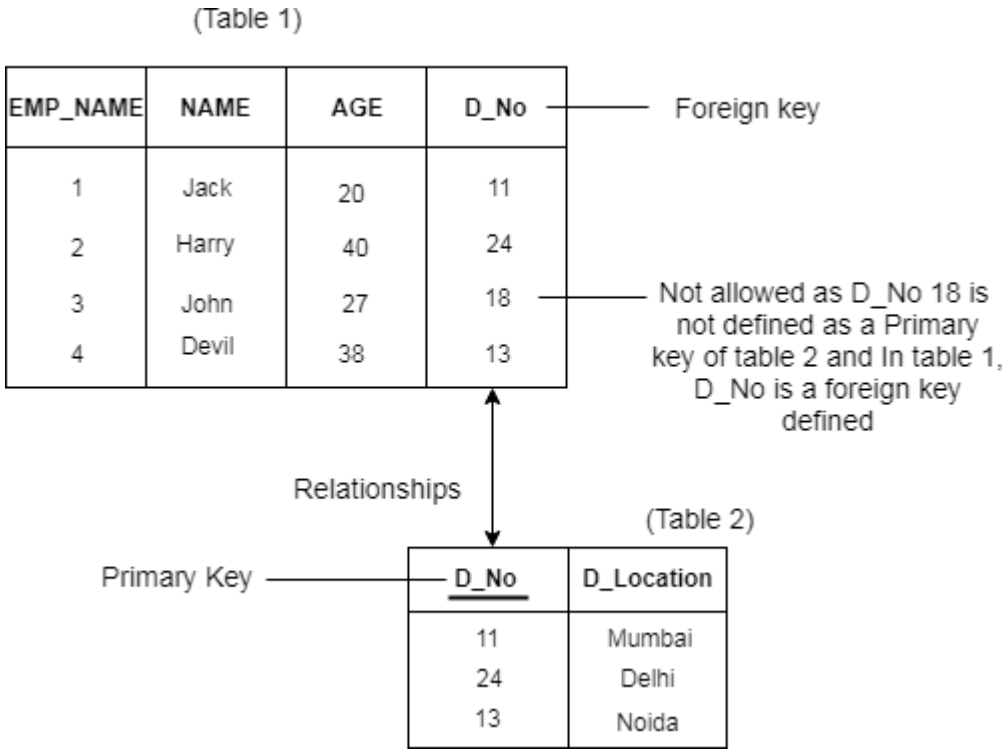
Example:

3.Referential Integrity Constraints:

Referential integrity constraint is specified between two tables.

- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2 then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:



3. Key constraints

- Keys are the entity set that is used to uniquely identify an entity within its entity set.
- An entity set can have multiple key but out of which one key will be primary key. A primary key can contain only unique and null value in the relational table.

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

Example:

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.

For 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 –

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

Notation – 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.

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

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**.

```
∏ author (Books) – ∏ 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}

```
σauthor = 'KORTH'(Books X Articles)
```

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

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.

For 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 (∀).

For 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 –

$\{ a_1, a_2, a_3, ..., a_n \mid P (a_1, a_2, a_3, ... ,a_n)\}$

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

For example –

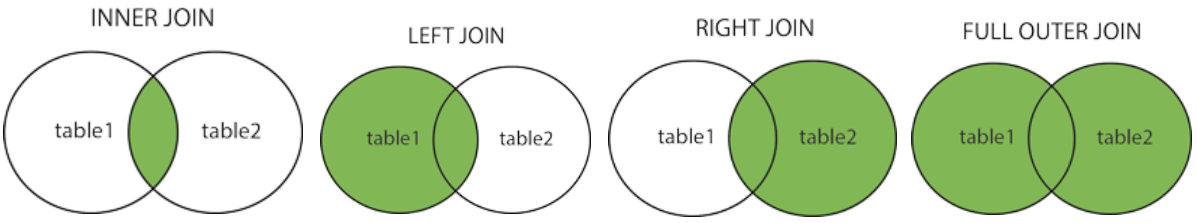
```
{< article, page, subject > |  ∈ KORTH ∧ subject = 'database'}
```

Output – Yields Article, Page, and Subject from the relation KORTH, where subject is database.

Join With types:

Different Types of JOINs(INNER) JOIN: Returns records that have matching values in both tables

- LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table.
- RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the left table.
- FULL (OUTER) JOIN: Return all records when there is a match in either left or right table.



Below is a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

And a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	AlfredsFutterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

SQL INNER JOIN Example

The following SQL statement selects all orders with customer information:

Example

```
SELECT Orders.OrderID,                    Customers.CustomerName
```

FROM Orders
INNER JOIN Customers ON Orders.CustomerID =

Customers.CustomerID; SQL LEFT JOIN Example

The following SQL statement will select all customers, and any orders they might have:

Example

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID =
Orders.CustomerID ORDER
BY Customers.CustomerName;
```

SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON
Customers.CustomerID=Orders.CustomerID ORDER BY
Customers.CustomerName;
```

SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they might have have placed:

Example

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID =
Employees.EmployeeID
ORDER BY Orders.OrderID
```

SQL

- SQL stands for Structure Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like: MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a numbers of ways, using English-like statements.

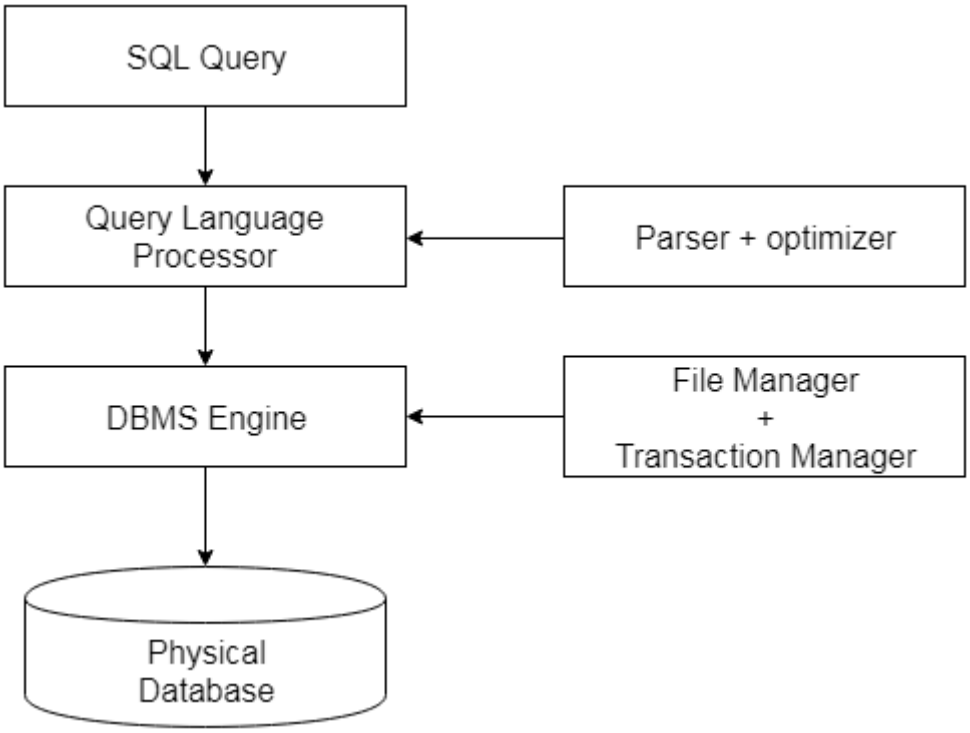
Rules:

SQL follows the following rules:

- Structure query language is not case sensitive. Generally keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

SQL process:

- When an SQL command is executing for any RDBMS then the system figure out the best way to carry out the request and SQL engine determines that how to interpret the task.
- In the process, various components are included. These components are optimization Engine, Query engine, Query dispatcher, classic etc.
- All the non-SQL queries are handled by the classic query engine but SQL query engine won't handle logical files.



Characteristics of SQL

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.

- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures and views.

Advantages of SQL

There are the following advantages of SQL:

High speed

Using the SQL queries, the user can quickly and efficiently retrieve large amount of records from a database.

No coding needed

In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

Well defined standards

Long established are used by the SQL databases that is being used by ISO and ANSI.

Portability

SQL can be used in laptop, PCs, server and even some of mobile phones.

Interactive language

SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

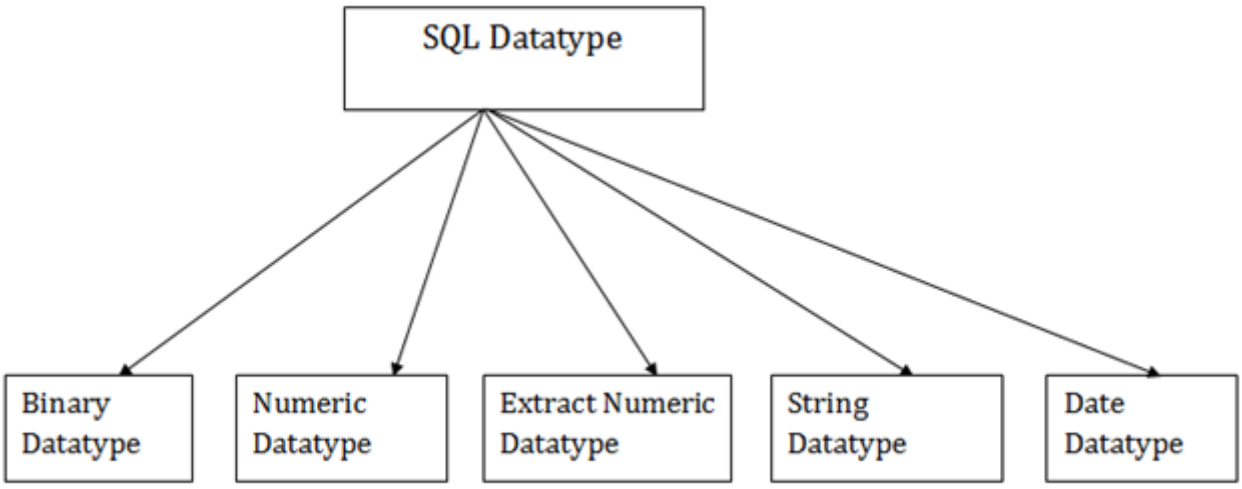
Multiple data view

Using the SQL language, the users can make different views of database structure.

SQL Datatype:

- SQL Data type is used to define the values that a column can contain.
- Every column is required to have a name and data type in the database table.

Datatype of SQL:



1. Binary Datatypes

There are Three types of binary Datatypes which are given below:

Data Type	Description
Binary	It has maximum length of 8000 bytes. It contains fixed-length binary data.
Varbinary	It has maximum length of 8000 bytes. It contains variable-length binary data.
Image	It has maximum length of 2,147,483,647 bytes. It contains variable-length binary data.

2. Approximate Numeric Datatype :

The subtypes are given below:

Data type	From	To	Description
Float	-1.79E + 308	1.79E + 308	It is used to specify a floating-point value e.g. 6.2, 2.9 etc.

Real	-3.40e + 38	3.40E + 38	It specifies a single precision floating point number

3. Exact Numeric Datatype

The subtypes are given below:

Data type	Description
Int	It is used to specify an integer value.
Smallint	It is used to specify small integer value.
Bit	It has the number of bit to store.
Decimal	It specifies a numeric values that can have a decimal number.
Numeric	It is used to specify a numeric value.

4. Character String Datatype

The subtypes are given below:

Data type	Description
Char	It has maximum length of 8000 characters. It contains Fixed-length non-unicode characters.
Varchar	It has maximum length of 8000 characters. It contains variable-length non-unicode characters.
text	It has maximum length of 2,147,483,647 characters. It contains variable-length non-unicode characters.

5. Date and time Datatypes

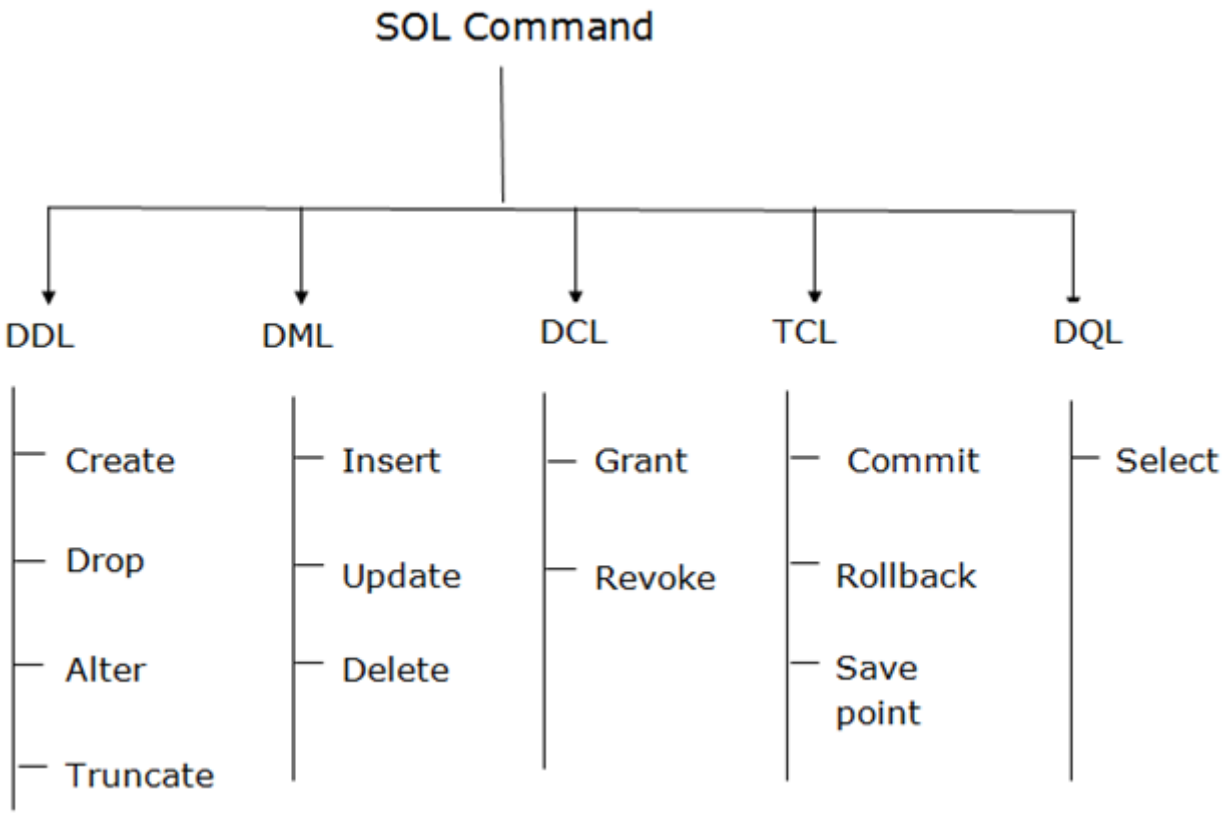
The subtypes are given below:

Datatype	Description
Date	It is used to store year, month and days value.
Time	It is used to store hour, minute and second values.
Timestamp	It stores year, month, day, hour, minute and second value.

SQL command

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions and queries of data.
- SQL can perform various tasks like: create table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Command:



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-committed 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 new table in the database.

Syntax:

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

Example:

```
1. 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

```
1. DROP TABLE ;
```

Example

```
1. DROP TABLE EMPLOYEE;
```

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

Syntax:

To add new column in table

```
1. ALTER TABLE table_name ADD column_name COLUMN-definition;
```

To modify existing column in table

```
1. ALTER TABLE MODIFY(COLUMN DEFINITION....);
```

EXAMPLE

```
1. ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
2. 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:

```
1. TRUNCATE TABLE table_name;
```

Example:

```
1. TRUNCATE TABLE EMPLOYEE;
```

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML are 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 a table.

Syntax:

```
1. INSERT INTO TABLE_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, .... valueN);
```

Or

```
1. INSERT INTO TABLE_NAME
2. VALUES (value1, value2, value3, .... valueN);
```

For example:

```
1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");
```

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

Syntax:

```
1. UPDATE table_name SET [column_name1= value1,...column_nameN = va
   lueN] [WHERE CONDITION]
```

For example:

```
1. UPDATE students
2. SET User_Name = 'Sonoo'
3. WHERE Student_Id = '3'
```

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

Syntax:

```
1. DELETE FROM table_name [WHERE condition];
```

For example:

```
1. DELETE FROM javatpoint
2. WHERE Author="Sonoo";
```

3. Data Control Language

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

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

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

Example

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

4. Transaction Control Language

TCL commands can only use with DML commands like: INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

```
1. COMMIT;
```

Example:

```
1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

```
1. ROLLBACK;
```

Example:

```
1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;
```

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

```
1. SAVEPOINT SAVEPOINT_NAME;
```

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

```
1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;
```

For example:

```
1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;
```

Consider the following relation. Worker (Worker_Id, First_Name, Last_Name, Salary, Joining_Date, Department), Bonus (Worker_Ref_Id, Bonus_Date, Bonus_Amount), Title (Worker_Ref_Id, Worker_Title, Affected_From). The Primary key is Rollno, Isbn. Write the query in Relational algebra of the following:

(a) Write An SQL Query That Fetches The Unique Values Of DEPARTMENT From Worker Table And Prints Its Length.

```
Select distinct length(DEPARTMENT) from Worker;
```

(b) Write An SQL Query To Print Details For Workers With The First Name As "Vipul" And "Satish" From Worker Table.

```
Select * from Worker where FIRST_NAME in ('Vipul','Satish');
```

(c) Write An SQL Query To Print Details Of The Workers Who Have Joined In Feb’2014.

```
Select * from Worker where year(JOINING_DATE) = 2014 and
month(JOINING_DATE) = 2;
```

(d) Write An SQL Query To Fetch The No. Of Workers For Each Department In The Descending Order.

```
SELECT DEPARTMENT, count(WORKER_ID) No_Of_Workers
FROM worker
GROUP BY DEPARTMENT
ORDER BY No_Of_Workers DESC;
```

(e) Write An SQL Query To Fetch Duplicate Records Having Matching Data In Some Fields Of A Table.

```
SELECT WORKER_TITLE, AFFECTED_FROM, COUNT(*)
FROM Title
GROUP BY WORKER_TITLE, AFFECTED_FROM
HAVING COUNT(*) > 1;
```

(f) Write An SQL Query To Fetch The Departments That Have Less Than Five People In It.

```
SELECT DEPARTMENT, COUNT(WORKER_ID) as 'Number of Workers' FROM Worker GROUP
BY DEPARTMENT HAVING COUNT(WORKER_ID) < 5;
```

Characteristics Of SQL & five aggregate Function:

Solution. Characteristics of SQL

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures and views.

The following are the most commonly used SQL aggregate functions:

- AVG – calculates the average of a set of values.
- COUNT – counts rows in a specified table or view.
- MIN – gets the minimum value in a set of values.
- MAX – gets the maximum value in a set of values.

- SUM – calculates the sum of values.

SQL aggregate function

examples COUNT function

example

To get the number of the products in the products table, you use the COUNT function as follows:

```
SELECTCOUNT(*)FROM products;
```

AVG function example

To calculate the average units in stock of the products, you use the AVG function as follows:

```
SELECT
AVG(unitsinstock)
FROMproducts;
```

MIN function example

To get the minimum units in stock of products in the products table, you use the MIN function as follows:

```
SELECTMIN(unitsinstock)FROMproducts;
```

MAX function example

To get the maximum units in stock of products in the products table, you use the MAX function as shown in the following query:

```
SELECT

MAX(unitsinstock)

FROM products;
```

Domain and Tuple Relational Calculus:

Solution. Relational Calculus in non-procedural query language and has no description about how the query will work or the data will b fetched. It only focusses on what to do, and not on how to do it.

Relational Calculus exists in

two forms: **Tuple Relational**

Calculus (TRC) Domain

Relational Calculus (DRC)

TupleRelationalCalculus (RC):

In tuple relational calculus, we

work on filtering tuples based

on the given condition.

condition. Syntax: { T |

Condition }

In this form of relational calculus, we define a tuple variable, specify the table(relation) name in which the tuple is to be searched for, along with a condition.

We can also specify column name using a . dot operator, with the tuple variable to only get a certain attribute(column) in result.

A tuple variable is nothing but a name, can be anything, generally we use a single alphabet for this, so let's say T is a tuple variable.

To specify the name of the relation(table) in which we want to look for

data, we do the following: Relation(T), where T is our tuple variable.

For example if our table is Student, we would put it as Student(T)

Then comes the condition part, to specify a condition applicable for a particluar attribute(column), we can use the . dot variable with the tuple variable to specify it, like in table Student, if we want to get data for students with age greater than 17, then, we can write it as,

T.age > 17, where T is our tuple variable.

Putting it all together, if we want to use Tuple Relational Calculus to fetch names of students, from table Student, with age greater than 17, then, for T being our tuple variable,

T.name | Student(T) AND T.age > 17

Domain Relational Calculus (DRC)

In domain relational calculus, filtering is done based on the domain of the attributes and not based on the tuple values.

Syntax: { c1, c2, c3, ..., cn | F(c1, c2, c3, ... ,cn)}

where, c1, c2... etc represents domain of attributes(columns) and F defines the formula including the condition for fetching the data.

For example,

$\{< name, age > \mid \in Student \wedge age > 17\}$

Again, the above query will return the names and ages of the students in

the table Student who are older than 17.

PL/SQL Trigger

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables

- Imposing security authorizations
- Preventing invalid transactions

Creating a trigger:

Syntax for creating trigger:

1. **CREATE** [OR REPLACE] **TRIGGER** trigger_name
2. {BEFORE | **AFTER** | **INSTEAD OF** }
3. {**INSERT** [OR] | **UPDATE** [OR] | **DELETE**}
4. [**OF** col_name]
5. **ON** table_name
6. [REFERENCING OLD **AS** o NEW **AS** n]
7. [**FOR** EACH ROW]
8. **WHEN** (condition)
9. **DECLARE**
10. Declaration-statements
11. **BEGIN**
12. Executable-statements
13. EXCEPTION
14. Exception-handling-statements
15. **END**;

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.

- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

PL/SQL Trigger Example

Let's take a simple example to demonstrate the trigger. In this example, we are using the following CUSTOMERS table:

Create table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000

4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create trigger:

Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
1. CREATE OR REPLACE TRIGGER display_salary_changes
2. BEFORE DELETE OR INSERT OR UPDATE ON customers
3. FOR EACH ROW
4. WHEN (NEW.ID > 0)
5. DECLARE
6.    sal_diff number;
7. BEGIN
8.    sal_diff := :NEW.salary  - :OLD.salary;
9.    dbms_output.put_line('Old salary: ' || :OLD.salary);
10.   dbms_output.put_line('New salary: ' || :NEW.salary);
11.   dbms_output.put_line('Salary difference: ' || sal_diff);
12.END;
13./
```

After the execution of the above code at SQL Prompt, it produces the following result.

Trigger created.

Check the salary difference by procedure:

Use the following code to get the old salary, new salary and salary difference after the trigger created.

```
1. DECLARE
2.    total_rows number(2);
3. BEGIN
4.    UPDATE  customers
5.    SET salary = salary + 5000;
6.    IF sql%notfound THEN
7.       dbms_output.put_line('no customers updated');
8.    ELSIF sql%found THEN
9.       total_rows := sql%rowcount;
10.      dbms_output.put_line( total_rows || ' customers updated ');
11.   END IF;
12.END;
13./
```

Output:

Old salary: 20000

New salary: 25000

Salary difference: 5000

Old salary: 22000

New salary: 27000

Salary difference: 5000

Old salary: 24000

New salary: 29000

Salary difference: 5000

Old salary: 26000

New salary: 31000

Salary difference: 5000

Old salary: 28000

New salary: 33000

Salary difference: 5000

Old salary: 30000

New salary: 35000

Salary difference: 5000

6 customers updated

Note: As many times you executed this code, the old and new both salary is incremented by 5000 and hence the salary difference is always 5000.

After the execution of above code again, you will get the following result.

Old salary: 25000

New salary: 30000

Salary difference: 5000

Old salary: 27000

New salary: 32000

Salary difference: 5000

Old salary: 29000

New salary: 34000

Salary difference: 5000

Old salary: 31000

New salary: 36000

Salary difference: 5000

Old salary: 33000

New salary: 38000

Salary difference: 5000

Old salary: 35000

New salary: 40000

Salary difference: 5000

6 customers updated

Important Points

Following are the two very important point and should be noted carefully.

- OLD and NEW references are used for record level triggers these are not avialable for table level triggers.

- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.

Cursors

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.
There are two types of cursors in PL/SQL :

1. Implicit cursors.
2. Explicit cursors.

Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed.

Cursor Attributes	
Name	**Description**
%FOUND	Returns TRUE if record was fetched successfully, FALSE otherwise.
%NOTFOUND	Returns TRUE if record was not fetched successfully, FALSE otherwise.
%ROWCOUNT	Returns number of records fetched from

	cursor at that point in time.
%ISOPEN	Returns TRUE if cursor is open, FALSE otherwise.

Implicit cursors :

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.
When you execute DML statements like DELETE, INSERT, UPDATE and SELECT statements, implicit statements are created to process these statements.
Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations. The cursor attributes available are %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.
For example, When you execute INSERT, UPDATE, or DELETE statements the cursor attributes tell us whether any rows are affected and how many have been affected. When a SELECT... INTO statement is executed in a PL/SQL Block, implicit cursor attributes can be used to find out whether any row has been returned by the SELECT statement. PL/SQL returns an error when no data is selected.
Consider the PL/SQL Stock that uses implicit cursor attributes as shown below:

Example:

```
DECLARE
   Eid  number(3);
 BEGIN
   UPDATE emp set eid=&eid where salary=&salary; eid:=sql%rowcount;
   IF SQL%found then
       dbms_output.put_line('success');
   ELSE
```

```
        dbms_output.put_line ( ' not' ) ;
    END IF;
        dbms_output.put_line( 'rowcount'||eid);
 END;
 /
```

Output:

Run SQL Command Line

```
SQL>START D://c.sql
success
PL/SQL Procedure successfully completed.
```

Explicit Cursors :

They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.
An explicit cursor is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. We can provide a suitable name for the cursor.

Syntax:

```
CURSOR cursor_name IS select_statement;
```

cursor _name -A suitable name for the cursor.
Select_statement - A select query which returns multiple rows.

How to use Explicit Cursor?

There are four steps in using an Explicit Cursor.

1. **DECLARE** the cursor in the declaration section

2. **OPEN** the cursor in the Execution Section.
3. **FETCH** the data from cursor into PL/SQL variables or records in the Execution Section.
4. **CLOSE** the cursor in the Execution Section before you end the PL/SQL Block.

Declaring a Cursor in the Declaration Section:

Syntax:

CURSOR CURSORNAME IS SELECT…….;

How to access an Explicit Cursor?

These are the three steps in accessing the cursor.

Open the cursor :

Syntax:

OPEN cursor_name;

Fetch the records in the cursor one at a time :

Syntax:

FETCH cursor_name INTO record_name;

OR

FETCH cursor_name INTO variable_list;

Close the cursor :

Syntax:

CLOSE cursor__name;

When a cursor is opened, the first row becomes the current row. When the data is fetched it is copied to the record or variables and the logical pointer moves to the next row and it becomes the current row. On every fetch statement, the

pointer moves to the next row. If you want to fetch after the last row, the program will throw an error. When there is more than one row in a cursor we can use loops along with explicit cursor attributes to fetch all the records.

Points to remember while fetching a row:

1. We can fetch the rows in a cursor to a PL/SQL Record or a list of variables created in the PL/SQL Block.
2. If you are fetching a cursor to a PL/SQL Record, the record should have the same structure as the cursor.
3. If you are fetching a cursor to a list of variables, the variables should be listed in the same order in the fetch statement as the columns are present in the cursor.

General Form of using an explicit cursor is:

Syntax:

```
DECLARE
  variables;
  records;
  create a cursor;
BEGIN
  OPEN cursor;
  FETCH cursor;
  process the records;
  CLOSE cursor;
END;
```

Example:

```
DECLARE
  CURSOR er IS select eid,name from emp order by name ;
  id emp.eid%type;
  ename emp.name%type;
BEGIN
  OPEN er;
  Loop
   FETCH er into id,ename;
   Exit when er%notfound;
   dbms_output.put_line (id || ename);
  end loop;
    close er;
END;
/
```

Output:

Run SQL Command Line

```
SQL>start D://ec.sql
1||parimal
2||preet
PL/SQL Procedure successfully completed.
```

Formated by Tuotiral Blocks

Cursor For Loop :

Oracle provides another loop-statements to control loops specifically for cursors. This statements is a variation of the basic FOR loop , and it is known as cursor for loops.

Syntax:

FOR VARIABLE IN CURSORNAME

LOOP

<EXECUTE COMMANDS>

END LOOP

Parameterized Cursors :

Oracle allows to pass parameters to cursors that can be used to provide condition with WHERE clause.If parameters are passed to curso, that cursor is called a **parameterized cursors.**

Syntax:

CURSOR CURSORNAME (VARIABLENAME DATATYPES) IS SELECT.....

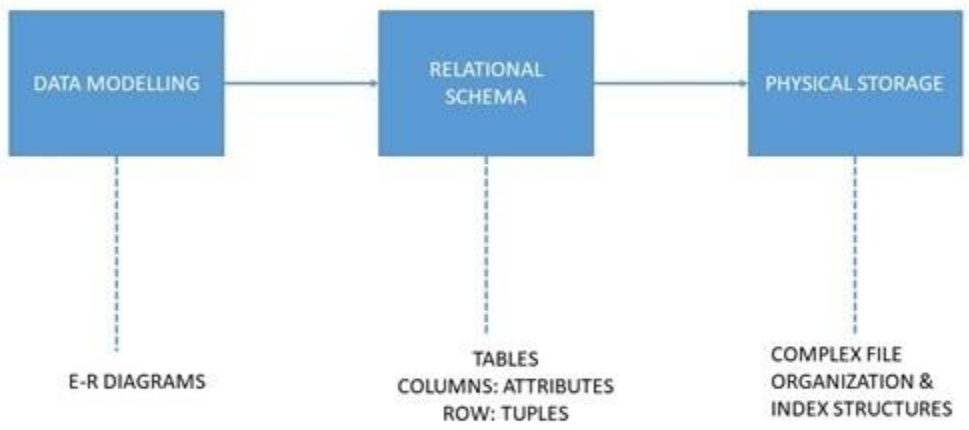
And, parameters canbe passed to cursor while opening it using syntax-

Syntax:

OPEN CURSORNAME (VALUE / VARIABLE / EXPRESSION);

UNIT-3 Relational Database Design

The relational data model was introduced by C. F. Codd in 1970. Currently, it is the most widely used data model. The relational data model describes the world as "a collection of inter-related relations (or tables)." A relational data model involves the use of data tables that collect groups of elements into relations. These models work based on the idea that each table setup will include a primary key or identifier. Other tables use that identifier to provide "relational" data links and results.



As mentioned, the primary key is a fundamental tool in creating and using relational data models. It must be unique for each member of a data set. It must be populated for all members. Inconsistencies can cause problems in how developers retrieve data. Other issues with relational database designs include excessive duplication of data, faulty or partial data, or improper links or associations between tables. A large part of routine database administration involves evaluating all the data sets in a database to make sure that they are consistently populated and will respond well to SQL or any other data retrieval method.

For example, a conventional database row would represent a tuple, which is a set of data that revolves around an instance or virtual object so that the primary key is its unique identifier. A column name in a data table is associated with an attribute, an identifier or feature that all parts of a data set have. These and other strict conventions help to provide database administrators and designers with standards for crafting relational database setups.

Database Design Objective

- **Eliminate Data Redundancy:** the same piece of data shall not be stored in more than one place. This is because duplicate data not only waste storage spaces but also easily lead to inconsistencies.
- **Ensure Data Integrity and Accuracy:** is the maintenance of, and the assurance of the accuracy and consistency of, data over its entire life-cycle, and is a critical aspect to the design, implementation, and usage of any system which stores, processes, or retrieves data.

The relational model has provided the basis for:

- Research on the theory of data/relationship/constraint
- Numerous database design methodologies
- The standard database access language called structured query language (SQL)
- Almost all modern commercial database management systems

Relational databases go together with the development of SQL. The simplicity of SQL - where even a novice can learn to perform basic queries in a short period of time - is a large part of the reason for the popularity of the relational model.

The two tables below relate to each other through the product code field. Any two tables can relate to each other simply by creating a field they have in common.

Table 1

Product_code	Description	Price
A416	Colour Pen	₹ 25.00
C923	Pencil box	₹ 45.00

Table 2

Invoice_code	Invoice_line	Product_code	Quantity
3804	1	A416	15
3804	2	C923	24

There are four stages of an RDM which are as follows −

- **Relations and attributes** − The various tables and attributes related to each table are identified. The tables represent entities, and the attributes represent the properties of the respective entities.
- **Primary keys** − The attribute or set of attributes that help in uniquely identifying a record is identified and assigned as the primary key.
- **Relationships** −The relationships between the various tables are established with the help of foreign keys. Foreign keys are attributes occurring in a table that are primary keys of another table. The types of relationships that can exist between the relations (tables) are One to one, One to many, and Many to many
- **Normalization** − This is the process of optimizing the database structure. Normalization simplifies the database design to avoid redundancy and confusion. The different normal forms are as follows:

1. First normal form 2. Second normal form 3. Third normal form 4. Boyce-Codd normal form 5. Fifth normal form

By applying a set of rules, a table is normalized into the above normal forms in a linearly progressive fashion. The efficiency of the design gets better with each higher degree of normalization.

Advantages of Relational Databases

The main advantages of relational databases are that they enable users to easily categorize and store data that can later be queried and filtered to extract specific information for reports. Relational databases are also easy to extend and aren't reliant on the physical organization. After the original database creation, a new data category can be added without all existing applications being modified.

Other Advantages

- **Accurate** − Data is stored just once, which eliminates data deduplication.
- **Flexible** − Complex queries are easy for users to carry out.
- **Collaborative** −Multiple users can access the same database.
- **Trusted** −Relational database models are mature and well-understood.
- **Secure** − Data in tables within relational database management systems (RDBMS) can be limited to allow access by only particular users.

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

1. X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependencies in DBMS:

1. Trivial functional dependency
2. Non-Trivial functional dependency

3. Multivalued functional dependency
4. Transitive functional dependency

1. Trivial Functional Dependency

In **Trivial Functional Dependency**, a dependent is always a subset of the determinant.
i.e. If **X → Y** and **Y is the subset of X**, then it is called trivial functional dependency

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, **{roll_no, name} → name** is a trivial functional dependency, since the dependent **name** is a subset of determinant set **{roll_no, name}**
Similarly, **roll_no → roll_no** is also an example of trivial functional dependency.

2. Non-trivial Functional Dependency

In **Non-trivial functional dependency**, the dependent is strictly not a subset of the determinant.
i.e. If **X → Y** and **Y is not a subset of X**, then it is called Non-trivial functional dependency.

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, **roll_no → name** is a non-trivial functional dependency, since the dependent **name** is **not a subset of** determinant **roll_no**
Similarly, **{roll_no, name} → age** is also a non-trivial functional dependency, since **age** is **not a subset of {roll_no, name}**

3. Multivalued Functional Dependency

In **Multivalued functional dependency**, entities of the dependent set are **not dependent on each other.**

i.e. If **a → {b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency.**
For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18
45	abc	19

Here, **roll_no → {name, age}** is a multivalued functional dependency, since the dependents **name** & **age** are **not dependent** on each other(i.e. **name → age** or **age → name doesn't exist !**)

5. ***Transitive Functional Dependency***

In transitive functional dependency, dependent is indirectly dependent on determinant.
i.e. If **a → b** & **b → c**, then according to axiom of transitivity, **a → c**. This is a **transitive functional dependency**
For example,

enrol_no	name	dept	building_no
42	abc	CO	4
43	pqr	EC	2
44	xyz	IT	1
45	abc	EC	2

Here, **enrol_no → dept** and **dept → building_no**,
Hence, according to the axiom of transitivity, **enrol_no → building_no** is a valid

functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

What is Normalization?

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

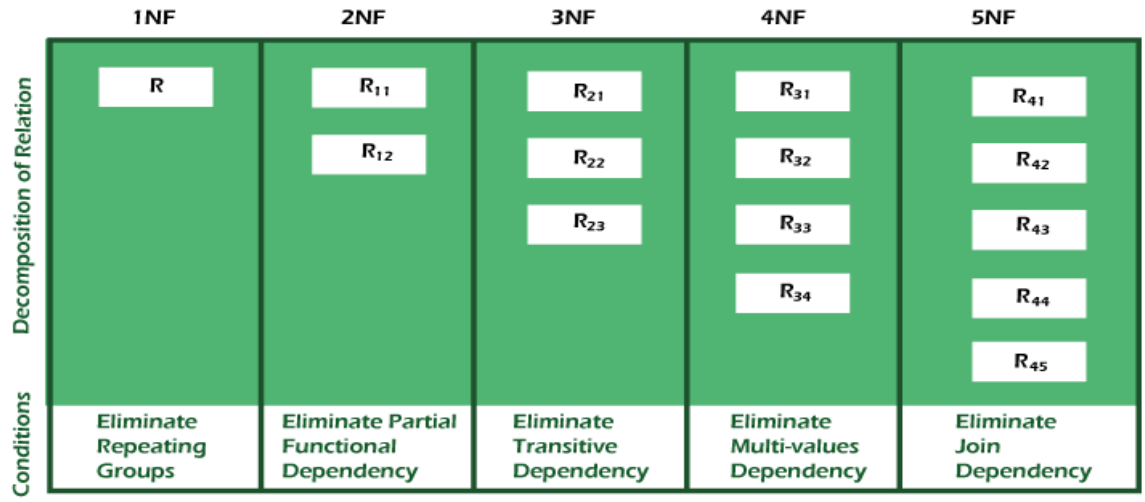
Data modification anomalies can be categorized into three types:

- **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

Following are the various types of Normal forms:



Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.

4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

Disadvantages of Normalization

- You cannot start building the database before knowing what the user needs.
- The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems.

Keys

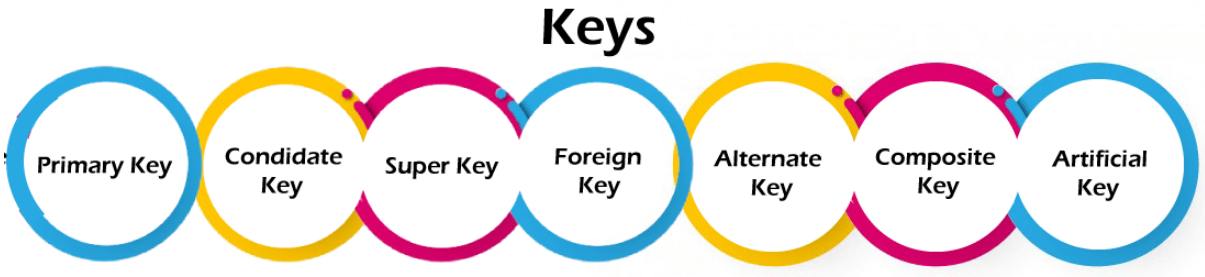
- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

For example, ID is used as a key in the Student table because it is unique for each student. In the PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.

STUDENT
ID
Name
Address
Course

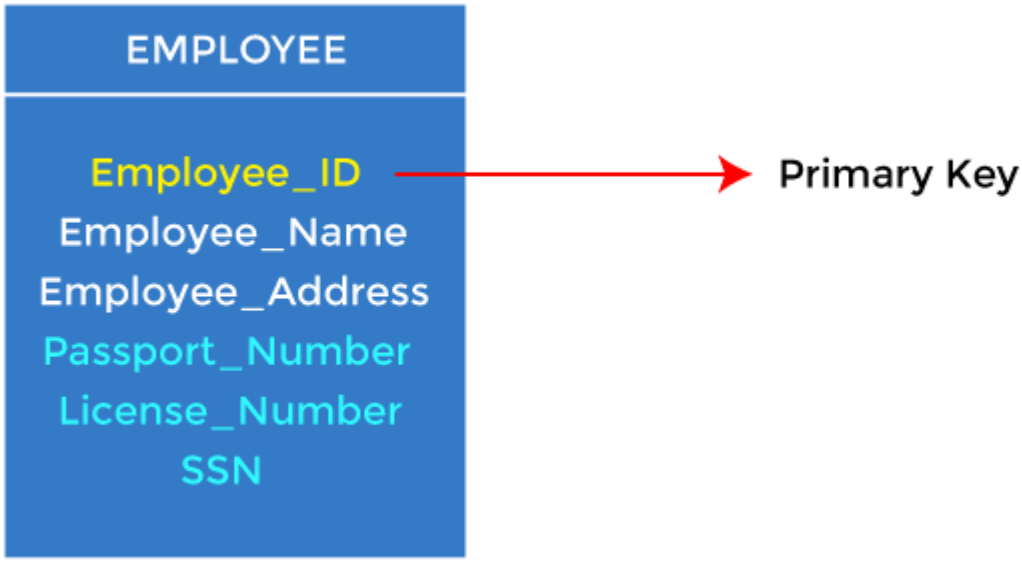
PERSON
Name
DOB
Passport, Number
License_Number
SSN

Types of keys:



1. Primary key

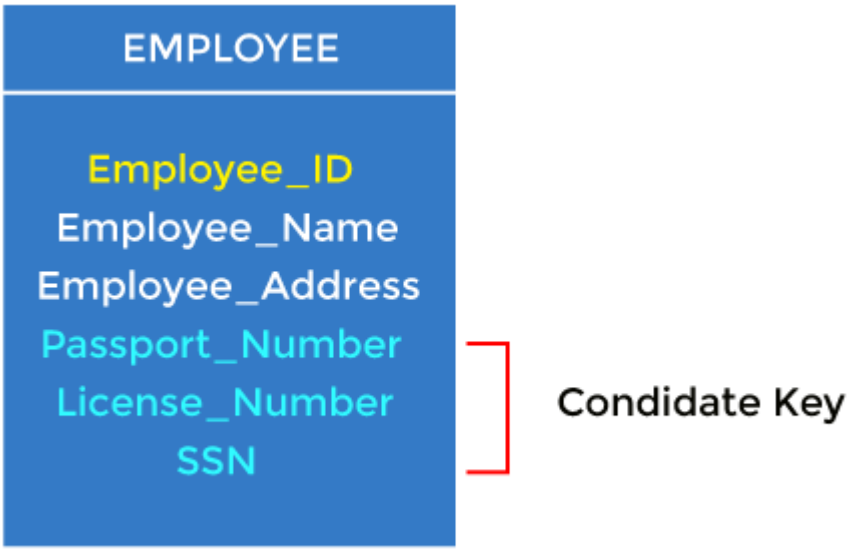
- It is the first key used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.
- In the EMPLOYEE table, ID can be the primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary keys since they are also unique.
- For each entity, the primary key selection is based on requirements and developers.



2. Candidate key

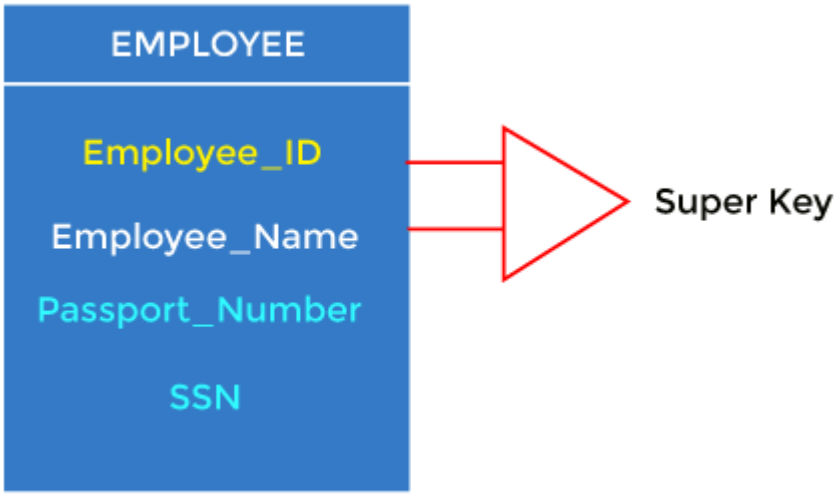
- A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
- Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. The rest of the attributes, like SSN, Passport_Number, License_Number, etc., are considered a candidate key.



3. Super Key

Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.

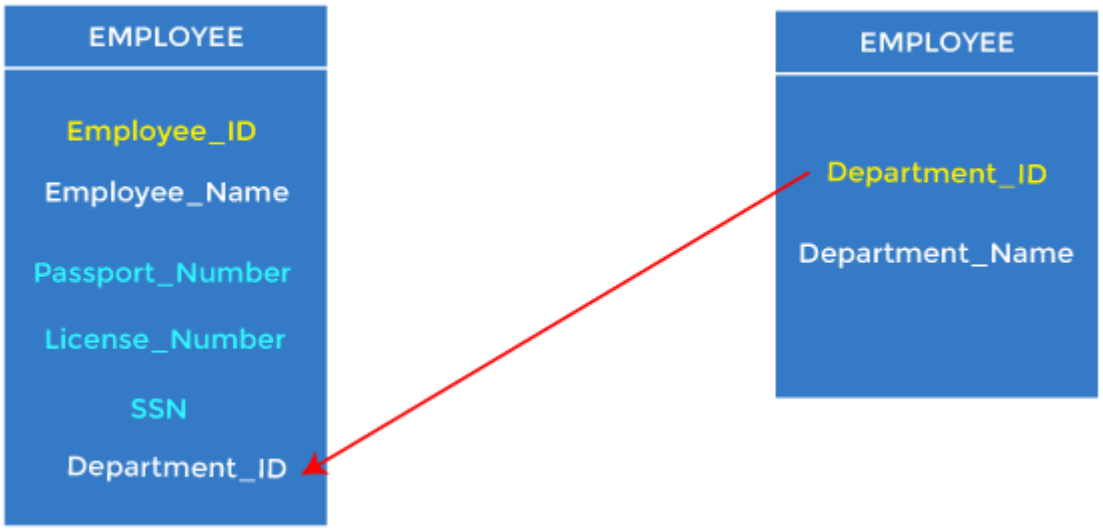


For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME), the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.

The super key would be EMPLOYEE-ID (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

4. Foreign key

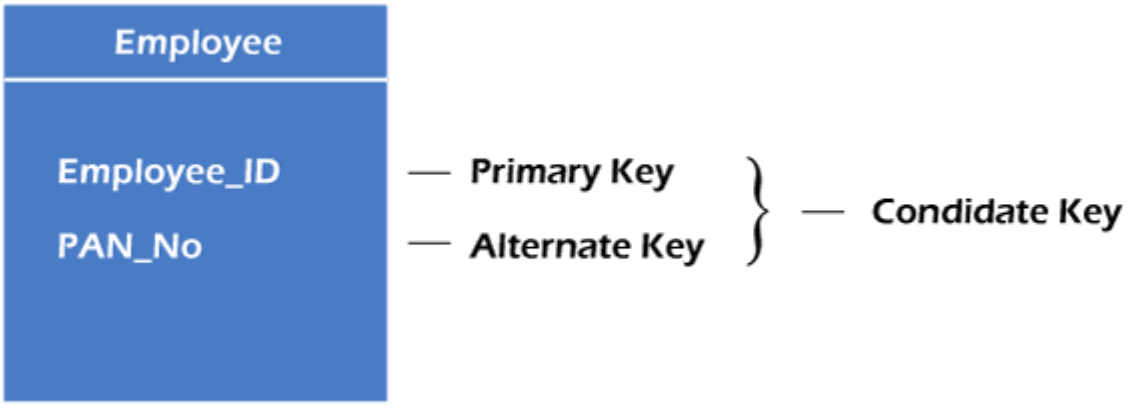
- Foreign keys are the column of the table used to point to the primary key of another table.
- Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id, as a new attribute in the EMPLOYEE table.
- In the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



5. Alternate key

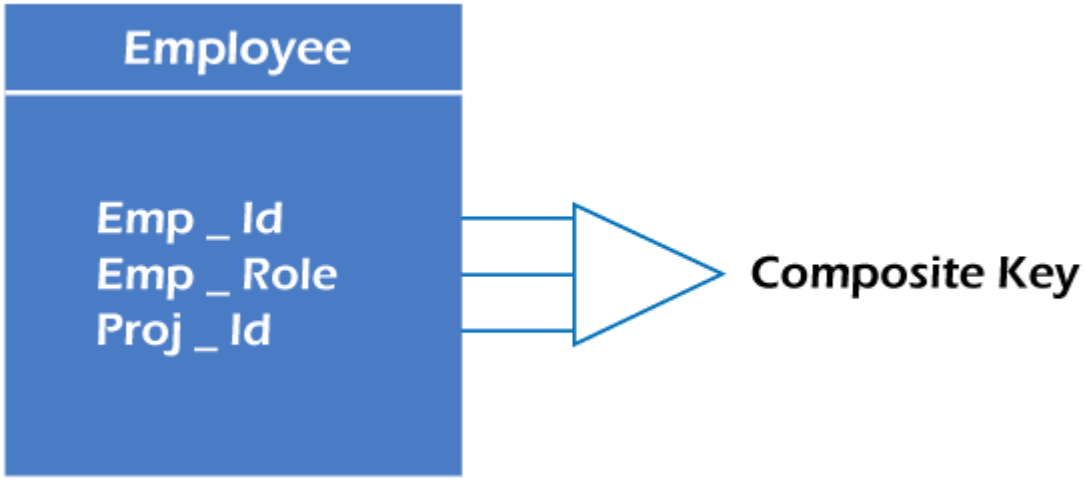
There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternate key. **In other words,** the total number of the alternate keys is the total number of candidate keys minus the primary key. The alternate key may or may not exist. If there is only one candidate key in a relation, it does not have an alternate key.

For example, employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. In this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as the Alternate key.

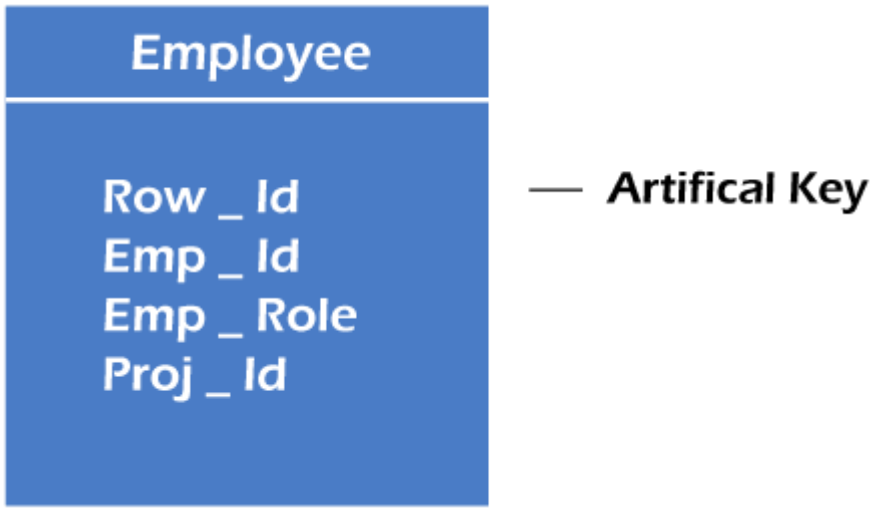


6. Composite key

Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.



For example, in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously. So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute.



7. Artificial key

The key created using arbitrarily assigned data are known as artificial keys. These keys are created when a primary key is large and complex and has no relationship with many other relations. The data values of the artificial keys are usually numbered in a serial order.

For example, the primary key, which is composed of Emp_ID, Emp_role, and Proj_ID, is large in employee relations. So it would be better to add a new virtual attribute to identify each tuple in the relation uniquely.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.

- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English

83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

   Example:

   EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

   Super key in the table above:

   1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so o n

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.

- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO

Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

lossless join decomposition

Lossless-join decomposition is a process in which a relation is decomposed into two or more relations. This property guarantees that the extra or less tuple generation problem does not occur and no information is lost from the original relation during the decomposition. It is also known as non-additive join decomposition.

When the sub relations combine again then the new relation must be the same as the original relation was before decomposition.

Consider a relation R if we decomposed it into sub-parts relation R1 and relation R2.

The decomposition is lossless when it satisfies the following statement −

- If we union the sub Relation R1 and R2 then it must contain all the attributes that are available in the original relation R before decomposition.
- Intersections of R1 and R2 cannot be Null. The sub relation must contain a common attribute. The common attribute must contain unique data.

The common attribute must be a super key of sub relations either R1 or R2.

Here,

R = (A, B, C)

R1 = (A, B)

R2 = (B, C)

The relation R has three attributes A, B, and C. The relation R is decomposed into two relation R1 and R2. . R1 and R2 both have 2-2 attributes.The common attributes are B.

The Value in Column B must be unique. if it contains a duplicate value then the Lossless-join decomposition is not possible.

Draw a table of Relation R with Raw Data −

R (A, B, C)

A	B	C
12	25	34
10	36	09
12	42	30

It decomposes into the two sub relations −

R1 (A, B)

A	B
12	25
10	36
12	42

R2 (B, C)

B	C
25	34
36	09
42	30

Now, we can check the first condition for Lossless-join decomposition.

The union of sub relation R1 and R2 is the same as relation R.

$R_1 U R_2 = R$

We get the following result −

A	B	C
12	25	34
10	36	09
12	42	30

The relation is the same as the original relation R. Hence, the above decomposition is Lossless-join decomposition.

Advantages of Lossless Join and Dependency Preserving Decomposition:

Improved Data Integrity: Lossless join and dependency preserving decomposition help to maintain the data integrity of the original relation by ensuring that all dependencies are preserved.
Reduced Data Redundancy: These techniques help to reduce data redundancy by breaking down a relation into smaller, more manageable relations.
Improved Query Performance: By breaking down a relation into smaller, more focused relations, query performance can be improved.
Easier Maintenance and Updates: The smaller, more focused relations are easier to maintain and update than the original relation, making it easier to modify the database schema and update the data.
Better Flexibility: Lossless join and dependency preserving decomposition can improve the flexibility of the database system by allowing for easier modification of the schema.

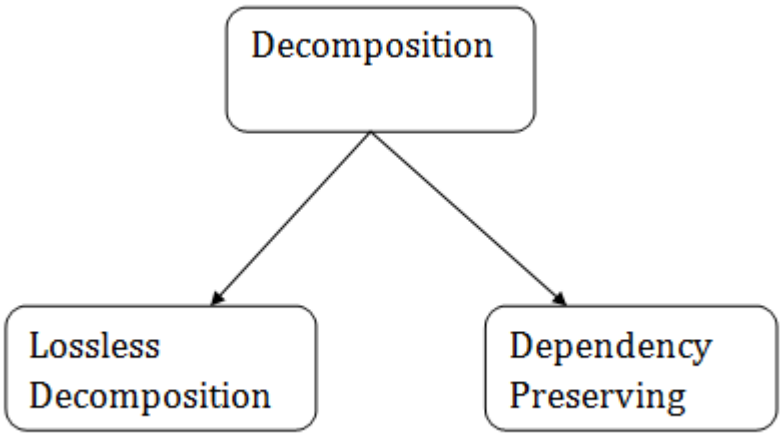
Disadvantages of Lossless Join and Dependency Preserving Decomposition:

Increased Complexity: Lossless join and dependency preserving decomposition can increase the complexity of the database system, making it harder to understand and manage.
Costly: Decomposing relations can be costly, especially if the database is large and complex. This can require additional resources, such as hardware and personnel.
Reduced Performance: Although query performance can be improved in some cases, in others, lossless join and dependency preserving decomposition can result in reduced query performance due to the need for additional join operations.
Limited Scalability: These techniques may not scale well in larger databases, as the number of smaller, focused relations can become unwieldy.

Relational Decomposition

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
- In a database, it breaks the table into multiple tables.
- If the relation has no proper decomposition, then it may lead to problems like loss of information.
- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Types of Decomposition



Lossless Decomposition

- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Example:

EMPLOYEE_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales

33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT table

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production

678	60	Testing

Now, when these two relations are joined on the common column "EMP_ID", then the resultant relation will look like:

Employee ⋈ Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Hence, the decomposition is Lossless join decomposition.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer

Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Domain-Key Normal Form

A relation is in DKNF when insertion or delete anomalies are not present in the database. Domain-Key Normal Form is the highest form of Normalization. The reason is that the insertion and updation anomalies are removed. The constraints are verified by the domain and key constraints.

A table is in Domain-Key normal form only if it is in 4NF, 3NF and other normal forms. It is based on constraints −

Domain Constraint

Values of an attribute had some set of values, for example, EmployeeID should be four digits long −

EmpID	EmpName	EmpAge
0921	Tom	33
0922	Jack	31

Key Constraint

An attribute or its combination is a candidate key

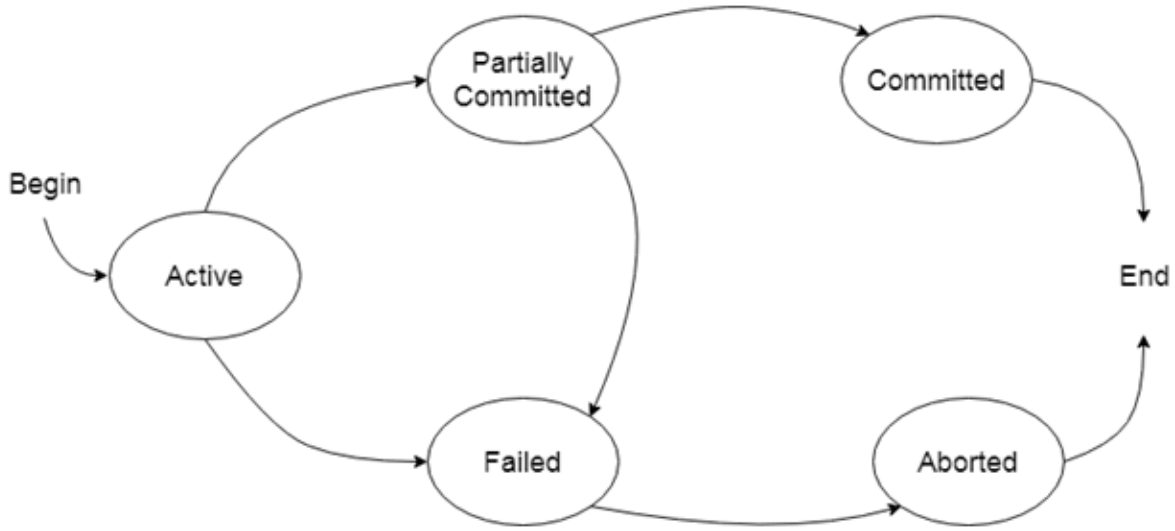
General Constraint

Predicate on the set of all relations.

Every constraint should be a logical sequence of the domain constraints and key constraints applied to the relation. The practical utility of DKNF is less.

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. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

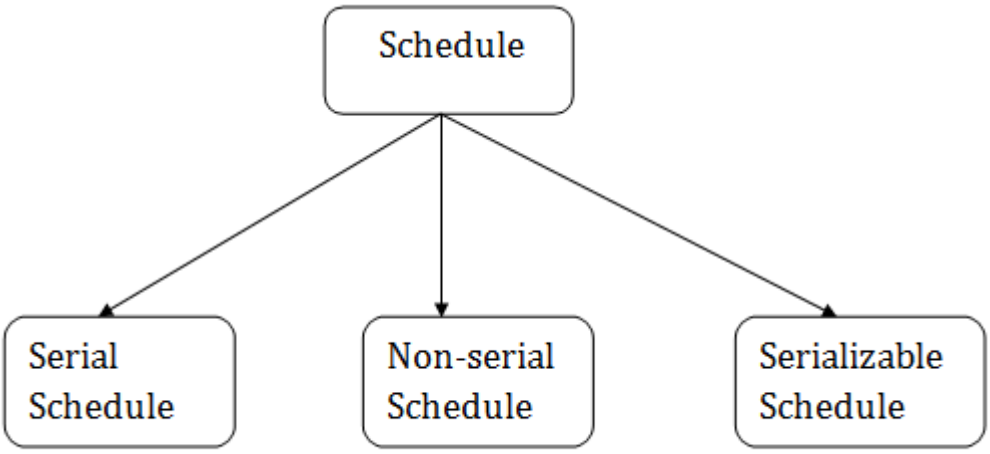
- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

- 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.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    1. Re-start the transaction
    2. Kill the transaction

Schedule

A series of operation from one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transaction.



1. Serial Schedule

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

1. Execute all the operations of T1 which was followed by all the operations of T2.
2. Execute all the operations of T1 which was followed by all the operations of T2.

- In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.
- In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

2. Non-serial Schedule

- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

3. Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

(a)

Time

T1	T2

read(A);
A := A – N;
write(A);
read(B);
B := B + N;
write(B);
	read(A);
	A := A + M;
	write(A);

Schedule A

(b)

Time

T1	T2

read(A);
A:= A - N;
write(A);
read(B);
B:= B +N;
write(B);
	read(A);
	A := A + M;
	write(A);

Schedule B

(c)

Time

T1	T2
read(A); A := A – N;	
	read(A); A := A + M;
write(A); read(B);	
	write(A);
B := B + N; write(B);	

Schedule C

(d)

Time

T1	T2

read(A);
A := A – N;
write(A);	

	read(A);
	A := A + M;
	write(A);

read(B);
B := B + N;
write(B);	

Schedule D

Here,

Schedule A and Schedule B are serial schedule.

Schedule C and Schedule D are Non-serial schedule.

Serializability in DBMS

The backbone of the most modern application is the form of DBMS. When we design the form properly, then it provides high-performance and relative storage solutions to our application. In this topic, we are going to explain the serializability concept and how this concept affects the DBMS deeply. We also understand the concept of serializability with some examples. Finally, we will conclude this topic with an example of the importance of serializability.

What is Serializability in DBMS?

In the field of computer science, serializability is a term that is a property of the system that describes how the different process operates the shared data. If the result given by the system is similar to the operation performed by the system, then in this situation, we call that system serializable. Here the cooperation of the system means there is no overlapping in the execution of the data. In DBMS, when the data is being written or read then, the DBMS can stop all the other processes from accessing the data.

In the MongoDB developer certificate, the DBMS uses various locking systems to allow the other processes while maintaining the integrity of the data. In MongoDB, the most restricted level for serializability is the employee can be restricted by two-phase locking or 2PL. In the first phase of the locking level, the data objects are locked before the execution of the operation. When the transaction has been accomplished, then the lock for the data object is released. This process guarantees that there is no conflict in operation and that all the transaction views the database as a conflict database.

The two-phase locking or 2PL system provides a strong guarantee for the conflict of the database.

It can reduce the decreased performance and then increase the overhead acquiring capacity and then release the lock of the data. As a result, the system allows the constraint serializability for better performance of the DBMS. This ensures that the final result is the same as some sequential execution and performs the improvement of the operation that is involved in the database.

Thus, serializability is the system's property that describes how the different process operates the shared data. In DBMS, the overall Serializable property is adopted by locking the data during the execution of other processes. Also, serializability ensures that the final result is equivalent to the sequential operation of the data.

What is a Serializable Schedule?

- In DBMS, the Serializable schedule is a property in which the read and write operation sequence does not disturb the serializability property. This property ensures that the

transaction is executed automatically with the other transaction. In DBMS, the order of the serializability must be the same as some serial schedules of the same transaction.

- In DBMS, there are several algorithms available to check the serializability of the database. One of the most important algorithms is the conflict serializability algorithm. The conflict serializability algorithm is the ability to check the potential of the conflict in the database. When the two transactions access the same data, this conflict occurs in the database. If there is no conflict, then there is guaranteed serializability in the database. However, if there is a conflict occurs, then there is a chance of serializability.
- Another algorithm that is used to check the serializability of the database is the DBMS algorithm. With the help of the DBMS algorithm, we can check the potential mutual dependencies between two transactions. When the two transactions give the correct output, then mutual dependencies exist. When there are no mutual dependencies, then there is a guaranteed serializable in the database. However, if there are mutual dependencies, then there will be a chance of serializable.
- We can also check the serializability of the database by using the precedence graph algorithm. A precedence relationship exists when one transaction must precede another transaction for the schedule to be valid. If there are no cycles, then the serializability of schedules in DBMS is guaranteed. However, if there are cycles, the schedule may or may not be serializable. To understand different algorithms comprehensively, take the MongoDB Administration certification and get expert analysis on the concept of serializability in DBMS.

Types of Serializability

In DBMS, all the transaction should be arranged in a particular order, even if all the transaction is concurrent. If all the transaction is not serializable, then it produces the incorrect result.

In DBMS, there are different types of serializable. Each type of serializable has some advantages and disadvantages. The two most common types of serializable are view serializability and conflict serializability.

1. Conflict Serializability

Conflict serializability is a type of conflict operation in serializability that operates the same data item that should be executed in a particular order and maintains the consistency of the database. In DBMS, each transaction has some unique value, and every transaction of the database is based on that unique value of the database.

This unique value ensures that no two operations having the same conflict value are executed concurrently. For example, let's consider two examples, i.e., the order table

and the customer table. One customer can have multiple orders, but each order only belongs to one customer. There is some condition for the conflict serializability of the database. These are as below.

- Both operations should have different transactions.
- Both transactions should have the same data item.
- There should be at least one write operation between the two operations.

If there are two transactions that are executed concurrently, one operation has to add the transaction of the first customer, and another operation has added by the second operation. This process ensures that there would be no inconsistency in the database.

2. View Serializability

View serializability is a type of operation in the serializable in which each transaction should produce some result and these results are the output of proper sequential execution of the data item. Unlike conflict serialized, the view serializability focuses on preventing inconsistency in the database. In DBMS, the view serializability provides the user to view the database in a conflicting way.

In DBMS, we should understand schedules S1 and S2 to understand view serializability better. These two schedules should be created with the help of two transactions T1 and T2. To maintain the equivalent of the transaction each schedule has to obey the three transactions. These three conditions are as follows.

- The first condition is each schedule has the same type of transaction. The meaning of this condition is that both schedules S1 and S2 must not have the same type of set of transactions. If one schedule has committed the transaction but does not match the transaction of another schedule, then the schedule is not equivalent to each other.
- The second condition is that both schedules should not have the same type of read or write operation. On the other hand, if schedule S1 has two write operations while schedule S2 has one write operation, we say that both schedules are not equivalent to each other. We may also say that there is no problem if the number of the read operation is different, but there must be the same number of the write operation in both schedules.
- The final and last condition is that both schedules should not have the same conflict. Order of execution of the same data item. For example, suppose the transaction of schedule S1 is T1, and the transaction of schedule S2 is T2. The transaction T1 writes the data item A, and the transaction T2 also writes the data item A. in this case, the schedule is not equivalent to each other. But if the schedule has the same number of

each write operation in the data item then we called the schedule equivalent to each other.

Testing of Serializability in DBMS with Examples

Serializability is a type of property of DBMS in which each transaction is executed independently and automatically, even though these transactions are executed concurrently. In other words, we can say that if there are several transactions executed concurrently, then the main work of the serializability function is to arrange these several transactions in a sequential manner.

For better understanding, let's explain these with an example. Suppose there are two users Sona and Archita. Each executes two transactions. Let's transactions T1 and T2 are executed by Sona, and T3 and T4 are executed by Archita. Suppose transaction T1 reads and writes the data item A, transaction T2 reads the data item B, transaction T3 reads and writes the data item C and transaction T4 reads the data item D. Lets the schedule the above transaction as below.

1. •
2. T1: Read A → Write A→ Read B → Write B`
3. •
4. `T2: Read B → Write B`
5. •
6. `T3: Read C → Write C→ Read D → Write D`
7. • `T4: Read D → Write D

Let's first discuss why these transactions are not serializable.

In order for a schedule to be considered serializable, it must first satisfy the conflict serializability property. In our example schedule above, notice that Transaction 1 (T1) and Transaction 2 (T2) read data item B before either writing it. This causes a conflict between T1 and T2 because they are both trying to read and write the same data item concurrently. Therefore, the given schedule does not conflict with serializability.

However, there is another type of serializability called view serializability which our example does satisfy. View serializability requires that if two transactions cannot see each other's updates (i.e., one transaction cannot see the effects of another concurrent transaction), the schedule is considered to view serializable. In our example, Transaction 2 (T2) cannot see any updates made by Transaction 4 (T4) because they do not share common data items. Therefore, the schedule is viewed as serializable.

It's important to note that conflict serializability is a stronger property than view serializability because it requires that all potential conflicts be resolved before any updates are made (i.e., each transaction must either read or write each data item before

any other transaction can write it). View serializability only requires that if two transactions cannot see each other's updates, then the schedule is view serializable & it doesn't matter whether or not there are potential conflicts between them.

All in all, both properties are necessary for ensuring correctness in concurrent transactions in a database management system.

Benefits of Serializability in DBMS

Below are the benefits of using the serializable in the database.

1. **Predictable execution:** In serializable, all the threads of the DBMS are executed at one time. There are no such surprises in the DBMS. In DBMS, all the variables are updated as expected, and there is no data loss or corruption.
2. **Easier to Reason about & Debug:** In DBMS all the threads are executed alone, so it is very easier to know about each thread of the database. This can make the debugging process very easy. So we don't have to worry about the concurrent process.
3. **Reduced Costs:** With the help of serializable property, we can reduce the cost of the hardware that is being used for the smooth operation of the database. It can also reduce the development cost of the software.
4. **Increased Performance:**In some cases, serializable executions can perform better than their non-serializable counterparts since they allow the developer to optimize their code for performance.

Conclusion

DBMS transactions must follow the ACID properties to be considered serializable. There are different types of serializability in DBMS, each with its own benefits and drawbacks. In most cases, selecting the right type of serializability will come down to a trade-off between performance and correctness.

Selecting the wrong type of serializability can lead to errors in your database that can be difficult to debug and fix. Hopefully, this guide has given you a better understanding of how Serializability in DBMS works and what kinds of serializability exist.

DBMS Concurrency Control

Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.

But before knowing about concurrency control, we should know about concurrent execution.

Concurrent Execution in DBMS

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.
- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution

In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems (W - W Conflict)

The problem occurs *when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent*.

For example:

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.

Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problems (W-R Conflict)

The dirty read problem occurs *when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.*

For example:

Consider two transactions T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.
- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem (W-R Conflict)

Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

For example:

Consider two transactions, T_X and T_Y, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.
- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Thus, in order to maintain consistency in the database and avoid such problems that take place in concurrent execution, management is needed, and that is where the concept of Concurrency Control comes into role.

Concurrency Control

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

The concurrency control protocols ensure the *atomicity, consistency, isolation, durability* and *serializability* of the concurrent execution of the database transactions. Therefore, these protocols are categorized as:

- Lock Based Concurrency Control Protocol
- Time Stamp Concurrency Control Protocol
- Validation Based Concurrency Control Protocol

DBMS Concurrency Control

Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.

But before knowing about concurrency control, we should know about concurrent execution.

Concurrent Execution in DBMS

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.
- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution

In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and

the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems (W - W Conflict)

The problem occurs *when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent.*

For example:

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).

- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.

Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problems (W-R Conflict)

The dirty read problem occurs *when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.*

For example:

Consider two transactions T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_X	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.

- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem (W-R Conflict)

Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

For example:

Consider two transactions, T_X and T_Y, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.
- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Thus, in order to maintain consistency in the database and avoid such problems that take place in concurrent execution, management is needed, and that is where the concept of Concurrency Control comes into role.

Concurrency Control

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

The concurrency control protocols ensure the *atomicity, consistency, isolation, durability* and *serializability* of the concurrent execution of the database transactions. Therefore, these protocols are categorized as:

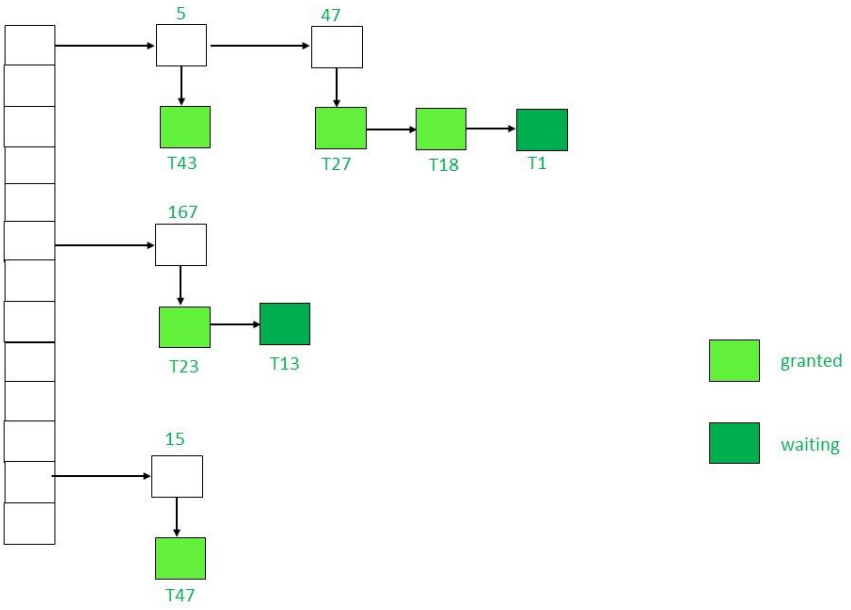
- Lock Based Concurrency Control Protocol
- Time Stamp Concurrency Control Protocol
- Validation Based Concurrency Control Protocol

Locking in DBMS

Locking protocols are used in database management systems as a means of concurrency control. Multiple transactions may request a lock on a data item simultaneously. Hence, we require a mechanism to manage the locking requests made by transactions. Such a mechanism is called as **Lock Manager**. It relies on the process of message passing where transactions and lock manager exchange messages to handle the locking and unlocking of data items. **Data structure used in Lock Manager** – The data structure required for implementation of locking is called as **Lock table**.

1. It is a hash table where name of data items are used as hashing index.
2. Each locked data item has a linked list associated with it.
3. Every node in the linked list represents the transaction which requested for lock, mode of lock requested (mutual/exclusive) and current status of the request (granted/waiting).
4. Every new lock request for the data item will be added in the end of linked list as a new node.
5. Collisions in hash table are handled by technique of separate chaining.

Consider the following example of lock table:



Explanation: In the above figure, the locked data items present in lock table are 5, 47, 167 and 15. The transactions which have requested for lock have been represented by a linked list shown below them using a downward arrow. Each node in linked list has the name of transaction which has requested the data item like T33, T1, T27 etc. The colour of node represents the status i.e. whether lock has been granted or waiting. Note that a collision has occurred for data item 5 and 47. It has been resolved by separate chaining where each data item belongs to a linked list. The data item is acting as header for linked list containing the locking request. **Working of Lock Manager –**

1. Initially the lock table is empty as no data item is locked.
2. Whenever lock manager receives a lock request from a transaction T_i on a particular data item Q_i following cases may arise:
    - If Q_i is not already locked, a linked list will be created and lock will be granted to the requesting transaction T_i.
    - If the data item is already locked, a new node will be added at the end of its linked list containing the information about request made by T_i.
3. If the lock mode requested by T_i is compatible with lock mode of transaction currently having the lock, T_i will acquire the lock too and status will be changed to ‘granted’. Else, status of T_i’s lock will be ‘waiting’.
4. If a transaction T_i wants to unlock the data item it is currently holding, it will send an unlock request to the lock manager. The lock manager will delete T_i’s node from this linked list. Lock will be granted to the next transaction in the list.
5. Sometimes transaction T_i may have to be aborted. In such a case all the waiting request made by T_i will be deleted from the linked lists

present in lock table. Once abortion is complete, locks held by T_i will also be released.

Advantages:

Data Consistency: Locking can help ensure data consistency by preventing multiple users from modifying the same data simultaneously. By controlling access to shared resources, locking can help prevent data conflicts and ensure that the database remains in a consistent state.
Isolation: Locking can ensure that transactions are executed in isolation from other transactions, preventing interference between transactions and reducing the risk of data inconsistencies.
Granularity: Locking can be implemented at different levels of granularity, allowing for more precise control over shared resources. For example, row-level locking can be used to lock individual rows in a table, while table-level locking can be used to lock entire tables.
Availability: Locking can help ensure the availability of shared resources by preventing users from monopolizing resources or causing resource starvation.

Disadvantages:

Overhead: Locking requires additional overhead, such as acquiring and releasing locks on shared resources. This overhead can lead to slower performance and increased resource consumption, particularly in systems with high levels of concurrency.
Deadlocks: Deadlocks can occur when two or more transactions are waiting for each other to release resources, causing a circular dependency that can prevent any of the transactions from completing. Deadlocks can be difficult to detect and resolve, and can result in reduced throughput and increased latency.
Reduced Concurrency: Locking can limit the number of users or applications that can access the database simultaneously. This can lead to reduced concurrency and slower performance in systems with high levels of concurrency.
Complexity: Implementing locking can be complex, particularly in distributed systems or in systems with complex transactional logic. This complexity can lead to increased development and maintenance costs.

Timestamp Ordering Protocol

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Recoverability of Schedule

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

The above table 1 shows a schedule which has two transactions. T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1. T2 should also be rollback because it

reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule.

Irrecoverable schedule: The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

The above table 2 shows a schedule with two transactions. Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1. T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is recoverable with cascade rollback.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

The above Table 3 shows a schedule with two transactions. Transaction T1 reads and write A and commits, and that value is read and written by T2. So this is a cascade less recoverable schedule.

Multiple Granularity

Let's start by understanding the meaning of granularity.

Granularity: It is the size of data item allowed to lock.

Multiple Granularity:

- It can be defined as hierarchically breaking up the database into blocks which can be locked.
- The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
- It maintains the track of what to lock and how to lock.
- It makes easy to decide either to lock a data item or to unlock a data item. This type of hierarchy can be graphically represented as a tree.

For example: Consider a tree which has four levels of nodes.

- The first level or higher level shows the entire database.
- The second level represents a node of type area. The higher level database consists of exactly these areas.
- The area consists of children nodes which are known as files. No file can be present in more than one area.
- Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.
- Hence, the levels of the tree starting from the top level are as follows:
    1. Database
    2. Area
    3. File
    4. Record

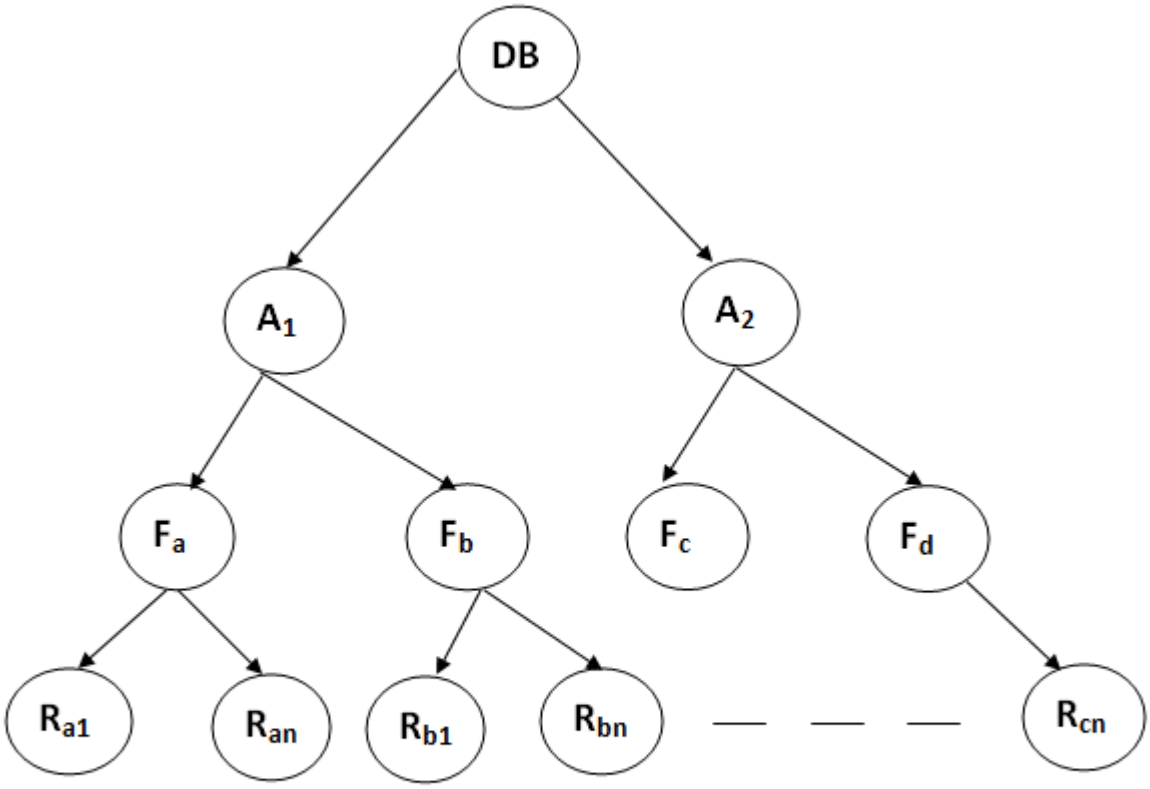


Figure: Multi Granularity tree Hierarchy

Deadlock Detection And Recovery

Deadlock detection and recovery is the process of detecting and resolving deadlocks in an operating system. A deadlock occurs when two or more processes are blocked, waiting for each other to release the resources they need. This can lead to a system-wide stall, where no process can make progress.

There are two main approaches to deadlock detection and recovery:

1. **Prevention**: The operating system takes steps to prevent deadlocks from occurring by ensuring that the system is always in a safe state, where deadlocks cannot occur. This is achieved through resource allocation algorithms such as the Banker's Algorithm.
2. **Detection and Recovery:** If deadlocks do occur, the operating system must detect and resolve them. Deadlock detection algorithms, such as the Wait-For Graph, are used to identify deadlocks, and recovery algorithms, such as the Rollback and Abort algorithm, are used to resolve them. The recovery algorithm releases the resources held by one or more processes, allowing the system to continue to make progress.

Difference Between Prevention and Detection/Recovery: Prevention aims to avoid deadlocks altogether by carefully managing resource allocation, while

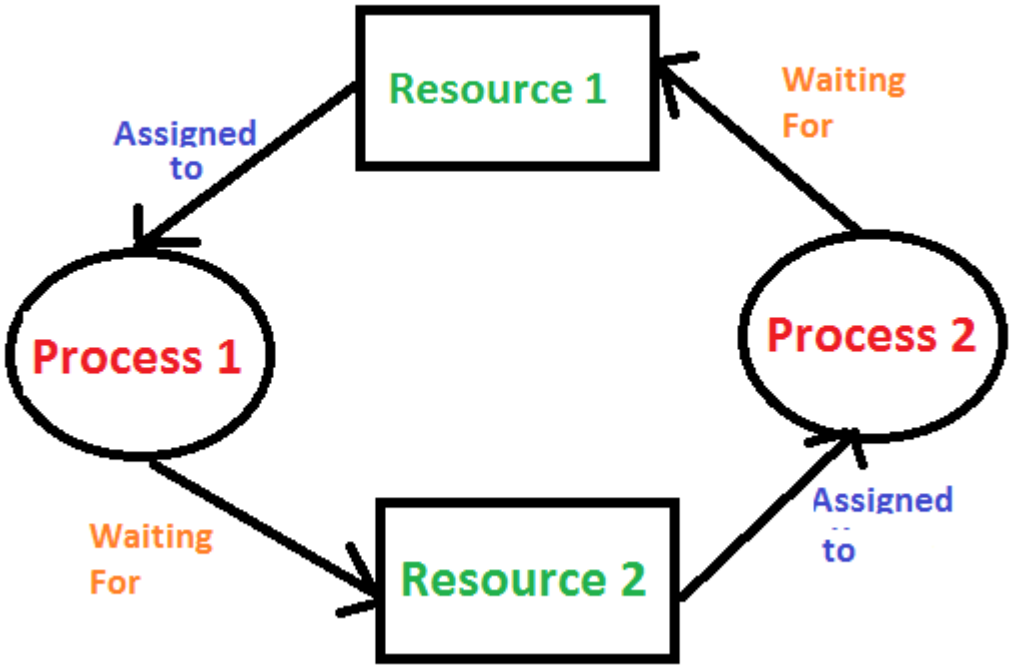
detection and recovery aim to identify and resolve deadlocks that have already occurred.
Deadlock detection and recovery is an important aspect of operating system design and management, as it affects the stability and performance of the system. The choice of deadlock detection and recovery approach depends on the specific requirements of the system and the trade-offs between performance, complexity, and risk tolerance. The operating system must balance these factors to ensure that deadlocks are effectively detected and resolved.

In the previous post, we discussed Deadlock Prevention and Avoidance. In this post, the Deadlock Detection and Recovery technique to handle deadlock is discussed.

Deadlock Detection :

1. If resources have a single instance –

In this case for Deadlock detection, we can run an algorithm to check for the cycle in the Resource Allocation Graph. The presence of a cycle in the graph is a sufficient condition for deadlock.



In the above diagram, resource 1 and resource 2 have single instances. There is a cycle R1 → P1 → R2 → P2. So, Deadlock is Confirmed.

2. If there are multiple instances of resources –

Detection of the cycle is necessary but not a sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.

3. Wait-For Graph Algorithm –

The Wait-For Graph Algorithm is a deadlock detection algorithm used to detect deadlocks in a system where resources can have multiple instances. The algorithm

works by constructing a Wait-For Graph, which is a directed graph that represents the dependencies between processes and resources.

Deadlock Recovery :
A traditional operating system such as Windows doesn't deal with deadlock recovery as it is a time and space-consuming process. Real-time operating systems use Deadlock recovery.

1. **Killing the process –**
   Killing all the processes involved in the deadlock. Killing process one by one. After killing each process check for deadlock again and keep repeating the process till the system recovers from deadlock. Killing all the processes one by one helps a system to break circular wait conditions.
2. **Resource Preemption –**
   Resources are preempted from the processes involved in the deadlock, and preempted resources are allocated to other processes so that there is a possibility of recovering the system from the deadlock. In this case, the system goes into starvation.
3. **Concurrency Control –** Concurrency control mechanisms are used to prevent data inconsistencies in systems with multiple concurrent processes. These mechanisms ensure that concurrent processes do not access the same data at the same time, which can lead to inconsistencies and errors. Deadlocks can occur in concurrent systems when two or more processes are blocked, waiting for each other to release the resources they need. This can result in a system-wide stall, where no process can make progress. Concurrency control mechanisms can help prevent deadlocks by managing access to shared resources and ensuring that concurrent processes do not interfere with each other.

ADVANTAGES OR DISADVANTAGES:

Advantages of Deadlock Detection and Recovery in Operating Systems:

1. **Improved System Stability:** Deadlocks can cause system-wide stalls, and detecting and resolving deadlocks can help to improve the stability of the system.
2. **Better Resource Utilization:** By detecting and resolving deadlocks, the operating system can ensure that resources are efficiently utilized and that the system remains responsive to user requests.
3. **Better System Design**: Deadlock detection and recovery algorithms can provide insight into the behavior of the system and the relationships between processes and resources, helping to inform and improve the design of the system.

Disadvantages of Deadlock Detection and Recovery in Operating Systems:

1. **Performance Overhead:** Deadlock detection and recovery algorithms can introduce a significant overhead in terms of performance, as the system

must regularly check for deadlocks and take appropriate action to resolve them.
2. **Complexity:** Deadlock detection and recovery algorithms can be complex to implement, especially if they use advanced techniques such as the Resource Allocation Graph or Timestamping.
3. **False Positives and Negatives:** Deadlock detection algorithms are not perfect and may produce false positives or negatives, indicating the presence of deadlocks when they do not exist or failing to detect deadlocks that do exist.
4. **Risk of Data Loss**: In some cases, recovery algorithms may require rolling back the state of one or more processes, leading to data loss or corruption.

Database Recovery Techniques in DBMS

Database recovery techniques are used in database management systems (DBMS) to restore a database to a consistent state after a failure or error has occurred. The main goal of recovery techniques is to ensure data integrity and consistency and prevent data loss. There are mainly two types of recovery techniques used in DBMS:

Rollback/Undo Recovery Technique: The rollback/undo recovery technique is based on the principle of backing out or undoing the effects of a transaction that has not completed successfully due to a system failure or error. This technique is accomplished by undoing the changes made by the transaction using the log records stored in the transaction log. The transaction log contains a record of all the transactions that have been performed on the database. The system uses the log records to undo the changes made by the failed transaction and restore the database to its previous state.
Commit/Redo Recovery Technique: The commit/redo recovery technique is based on the principle of reapplying the changes made by a transaction that has been completed successfully to the database. This technique is accomplished by using the log records stored in the transaction log to redo the changes made by the transaction that was in progress at the time of the failure or error. The system uses the log records to reapply the changes made by the transaction and restore the database to its most recent consistent state.
In addition to these two techniques, there is also a third technique called checkpoint recovery. Checkpoint recovery is a technique used to reduce the recovery time by periodically saving the state of the database in a checkpoint file. In the event of a failure, the system can use the checkpoint file to restore the database to the most recent consistent state before the failure occurred, rather than going through the entire log to recover the database.

Overall, recovery techniques are essential to ensure data consistency and availability in DBMS, and each technique has its own advantages and limitations that must be considered in the design of a recovery system

Database systems, like any other computer system, are subject to failures but the data stored in them must be available as and when required. When a database fails it must possess the facilities for fast recovery. It must also have atomicity i.e. either transaction are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database. There are both automatic and non-automatic ways for both, backing up of data and recovery from any failure situations. The techniques used to recover the lost data due to system crashes, transaction errors, viruses, catastrophic failure, incorrect commands execution, etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred update and immediate update or backing up data can be used. Recovery techniques are heavily dependent upon the existence of a special file known as a **system log**. It contains information about the start and end of each transaction and any updates which occur during the **transaction**. The log keeps track of all transaction operations that affect the values of database items. This information is needed to recover from transaction failure.

- The log is kept on disk start_transaction(T): This log entry records that transaction T starts the execution.
- read_item(T, X): This log entry records that transaction T reads the value of database item X.
- write_item(T, X, old_value, new_value): This log entry records that transaction T changes the value of the database item X from old_value to new_value. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X.
- commit(T): This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database.
- abort(T): This records that transaction T has been aborted.
- checkpoint: Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in a consistent state, and all the transactions were committed.

Stored Procedure

A **stored procedure** is a group of pre-compiled SQL statements (prepared SQL code) that can be reused again and again.

They can be used to perform a wide range of database operations such as inserting, updating, or deleting data, generating reports, and performing complex calculations. Stored procedures are very useful because they allow you to encapsulate (bundle) a set of SQL statements as a single unit and execute them repeatedly with different parameters, making it easy to manage and reuse code.

Procedures have similar structure as functions; they accept parameters and perform operations when we call them; but the difference between them is that SQL stored procedures are simpler to write or create, whereas functions have a more rigid structure and support fewer clauses and functionality.

Syntax

The basic syntax to create a stored procedure in SQL is as follows −

```
CREATE PROCEDURE procedure_name
  @parameter1 datatype,
  @parameter2 datatype
AS
BEGIN
  -- SQL statements to be executed
END
```

Where,

- The CREATE PROCEDURE statement is used to create the procedure. After creating the procedure, we can define any input parameters that the procedure may require. These parameters are preceded by the '@' symbol and followed by their respective data types.
- The AS keyword is used to begin the procedure definition. The SQL statements that make up the procedure are placed between the BEGIN and END keywords.

Creating a Procedure

We can create a stored procedure using the CREATE PROCEDURE statement in SQL. Following are the simple steps for creating a stored procedure −

- Choose a name for the procedure.
- Write the SQL code for the procedure, including to create the procedure in SQL Server.
- We can then test the stored procedure by executing it with different input parameters.

Example

To understand it better let us consider the CUSTOMERS table which contains the personal details of customers including their name, age, address and salary etc. as shown below −

```
CREATE TABLE CUSTOMERS (
   ID INT NOT NULL,
   NAME VARCHAR (20) NOT NULL,
   AGE INT NOT NULL,
   ADDRESS CHAR (25),
   SALARY DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

Now insert values into this table using the INSERT statement as follows −

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );


INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
```

```
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );


INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );


INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );


INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );


INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );


INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

The table will be created as −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Now, let's look at a simple example of creating a stored procedure that takes an input parameter and returns a result set.

In the following query, we are trying to first create the stored procedure with the name ‘GetCustomerInfo’. We then provide it with a single input parameter called @CutomerAge. The stored procedure then selects all records from the ‘CUSTOMERS’ table where the CutomerAge matches the input parameter.

```
CREATE PROCEDURE GetCustomerInfo

   @CutomerAge INT

AS

BEGIN
```

```
   SELECT * FROM CUSTOMERS

   WHERE AGE = @CutomerAge

END
```

Output

This would produce the following result −

```
Commands completed successfully.
```

Verification

To verify the changes, once we have created the stored procedure, we can test it by executing it with different input parameters as shown in the query below −

```
EXEC GetCustomerInfo @CutomerAge = 25
```

This will return all columns from the CUSTOMERS table where the cutomer's age is 25.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
+----+----------+-----+-----------+----------+
```

SQL Trigger

Trigger: A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

```
create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]
```

Explanation of syntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.

3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]: This provides the operation to be performed as trigger is fired

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run.
AFTER triggers run the trigger action after the triggering statement is run.

Example:

Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and percentage of specified marks is automatically inserted whenever a record is insert.
Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.

Suppose the database Schema –

```
mysql> desc Student;
```

```
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

SQL Trigger to problem statement.

```
create trigger stud_marks
before INSERT
on
Student
for each row
set Student.total = Student.subj1 + Student.subj2 + Student.subj3,
Student.per = Student.total * 60 / 100;
```

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. i.e.,

```
mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);
Query OK, 1 row affected (0.09 sec)


mysql> select * from Student;
+-----+-------+-------+-------+-------+-------+------+
| tid | name  | subj1 | subj2 | subj3 | total | per  |
+-----+-------+-------+-------+-------+-------+------+
| 100 | ABCDE |    20 |    20 |    20 |    60 |   36 |
+-----+-------+-------+-------+-------+-------+------+
1 row in set (0.00 sec)
```

Storage System in DBMS

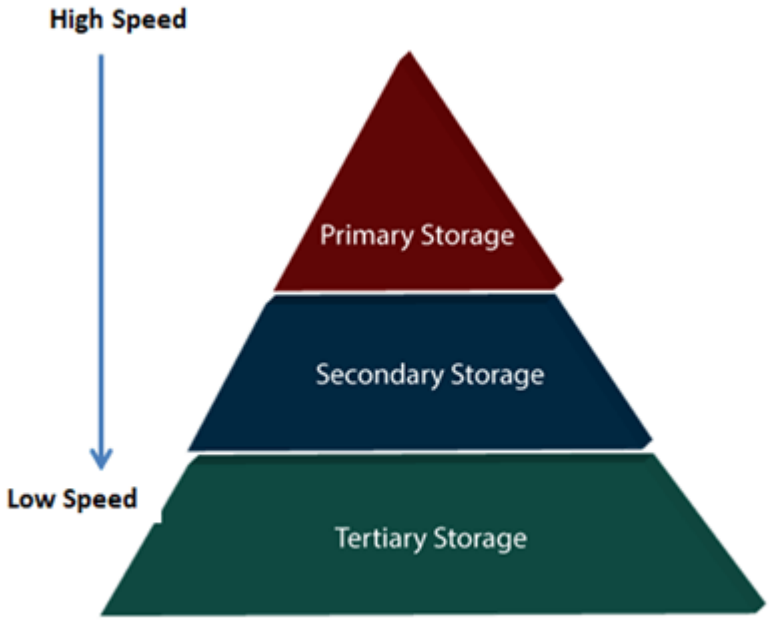
A database system provides an ultimate view of the stored data. However, data in the form of bits, bytes get stored in different storage devices.

In this section, we will take an overview of various types of storage devices that are used for accessing and storing data.

Types of Data Storage

For storing the data, there are different types of storage options available. These storage types differ from one another as per the speed and accessibility. There are the following types of storage devices used for storing the data:

- Primary Storage
- Secondary Storage
- Tertiary Storage



Primary Storage

It is the primary area that offers quick access to the stored data. We also know the primary storage as volatile storage. It is because this type of memory does not permanently store the data. As soon as the system leads to a power cut or a crash, the data also get lost. Main memory and cache are the types of primary storage.

- **Main Memory:** It is the one that is responsible for operating the data that is available by the storage medium. The main memory handles each instruction of a computer machine. This type of memory can store gigabytes of data on a system but is small enough to carry the entire database. At last, the main memory loses the whole content if the system shuts down because of power failure or other reasons.

1. **Cache:** It is one of the costly storage media. On the other hand, it is the fastest one. A cache is a tiny storage media which is maintained by the computer hardware usually. While designing the algorithms and query processors for the data structures, the designers keep concern on the cache effects.

Secondary Storage

Secondary storage is also called as Online storage. It is the storage area that allows the user to save and store data permanently. This type of memory does not lose the data due to any power failure or system crash. That's why we also call it non-volatile storage.

There are some commonly described secondary storage media which are available in almost every type of computer system:

- **Flash Memory:** A flash memory stores data in USB (Universal Serial Bus) keys which are further plugged into the USB slots of a computer system. These USB keys help transfer data to a computer system, but it varies in size limits. Unlike the main memory, it is possible to get back the stored data which may be lost due to a power cut or other reasons. This type of memory storage is most commonly used in the server systems for caching the frequently used data. This leads the systems towards high performance and is capable of storing large amounts of databases than the main memory.
- **Magnetic Disk Storage:** This type of storage media is also known as online storage media. A magnetic disk is used for storing the data for a long time. It is capable of storing an entire database. It is the responsibility of the computer system to make availability of the data from a disk to the main memory for further accessing. Also, if the system performs any operation over the data, the modified data should be written back to the disk. The tremendous capability of a magnetic disk is that it does not affect the data due to a system crash or failure, but a disk failure can easily ruin as well as destroy the stored data.

Tertiary Storage

It is the storage type that is external from the computer system. It has the slowest speed. But it is capable of storing a large amount of data. It is also known as Offline

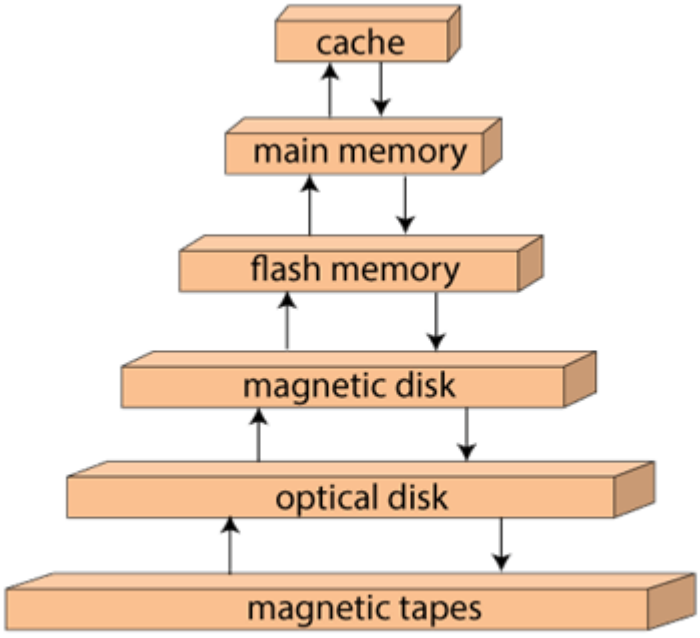
storage. Tertiary storage is generally used for data backup. There are following tertiary storage devices available:

- **Optical Storage:** An optical storage can store megabytes or gigabytes of data. A Compact Disk (CD) can store 700 megabytes of data with a playtime of around 80 minutes. On the other hand, a Digital Video Disk or a DVD can store 4.7 or 8.5 gigabytes of data on each side of the disk.
- **Tape Storage:** It is the cheapest storage medium than disks. Generally, tapes are used for archiving or backing up the data. It provides slow access to data as it accesses data sequentially from the start. Thus, tape storage is also known as sequential-access storage. Disk storage is known as direct-access storage as we can directly access the data from any location on disk.

Storage Hierarchy

Besides the above, various other storage devices reside in the computer system. These storage media are organized on the basis of data accessing speed, cost per unit of data to buy the medium, and by medium's reliability. Thus, we can create a hierarchy of storage media on the basis of its cost and speed.

Thus, on arranging the above-described storage media in a hierarchy according to its speed and cost, we conclude the below-described image:



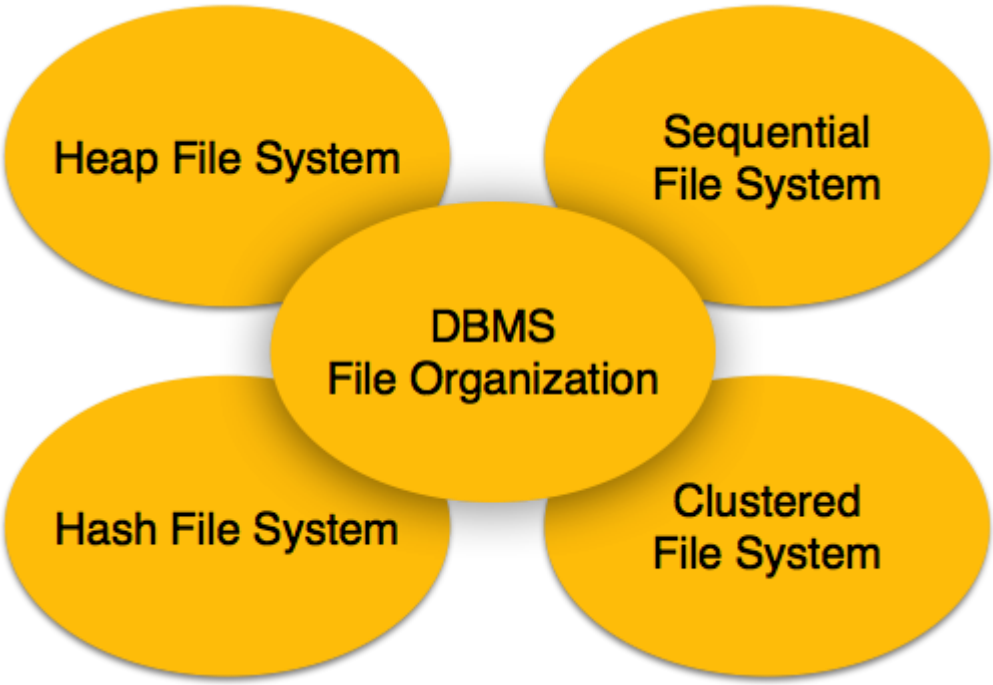
Storage device hierarchy

In the image, the higher levels are expensive but fast. On moving down, the cost per bit is decreasing, and the access time is increasing. Also, the storage media from the

main memory to up represents the volatile nature, and below the main memory, all are non-volatile devices.

File Organization

File Organization defines how file records are mapped onto disk blocks. We have four types of File Organization to organize file records –



Heap File Organization

When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area. It is the responsibility of the software to manage the records. Heap File does not support any ordering, sequencing, or indexing on its own.

Sequential File Organization

Every file record contains a data field (attribute) to uniquely identify that record. In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key. Practically, it is not possible to store all the records sequentially in physical form.

Hash File Organization

Hash File Organization uses Hash function computation on some fields of the records. The output of the hash function determines the location of disk block where the records are to be placed.

Clustered File Organization

Clustered file organization is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

File Operations

Operations on database files can be broadly classified into two categories −

- **Update Operations**
- **Retrieval Operations**

Update operations change the data values by insertion, deletion, or update. Retrieval operations, on the other hand, do not alter the data but retrieve them after optional conditional filtering. In both types of operations, selection plays a significant role. Other than creation and deletion of a file, there could be several operations, which can be done on files.

- **Open** − A file can be opened in one of the two modes, **read mode** or **write mode**. In read mode, the operating system does not allow anyone to alter data. In other words, data is read only. Files opened in read mode can be shared among several entities. Write mode allows data modification. Files opened in write mode can be read but cannot be shared.
- **Locate** − Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find (seek) operation, it can be moved forward or backward.
- **Read** − By default, when files are opened in read mode, the file pointer points to the beginning of the file. There are options where the user can tell the operating system where to locate the file pointer at the time of opening a file. The very next data to the file pointer is read.
- **Write** − User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification. The file pointer can be located at the time of opening or can be dynamically changed if the operating system allows to do so.
- **Close** − This is the most important operation from the operating system's point of view. When a request to close a file is generated, the operating system
    - removes all the locks (if in shared mode),
    - saves the data (if altered) to the secondary storage media, and
    - releases all the buffers and file handlers associated with the file.

The organization of data inside a file plays a major role here. The process to locate the file pointer to a desired record inside a file various based on whether the records are arranged sequentially or clustered.

Indexing

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types −

- **Primary Index** − Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Secondary Index** − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- **Clustering Index** − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Ordered Indexing is of two types −

- Dense Index
- Sparse Index

Dense Index

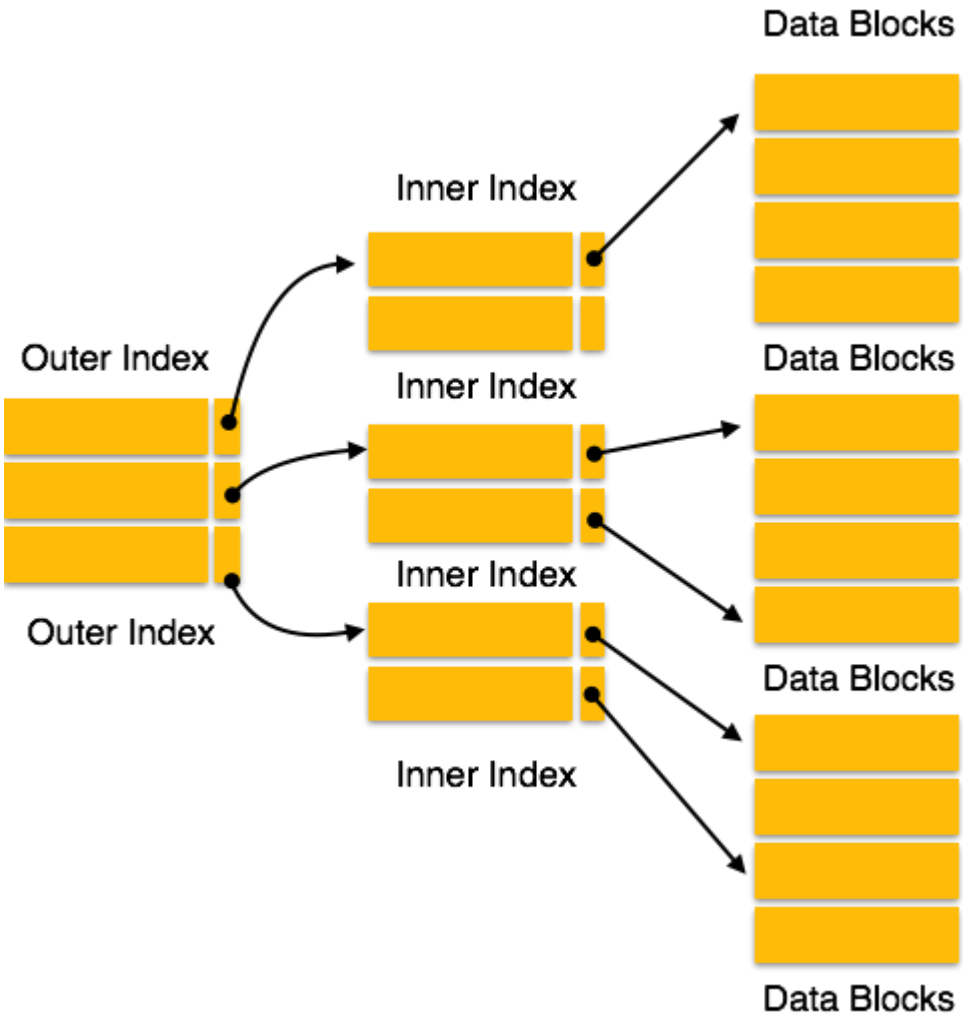
In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.

Sparse Index

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.

Multilevel Index

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

B tree vs B+ tree

Before understanding **B tree** and **B+ tree** differences, we should know the B tree and B+ tree separately.

What is the B tree?

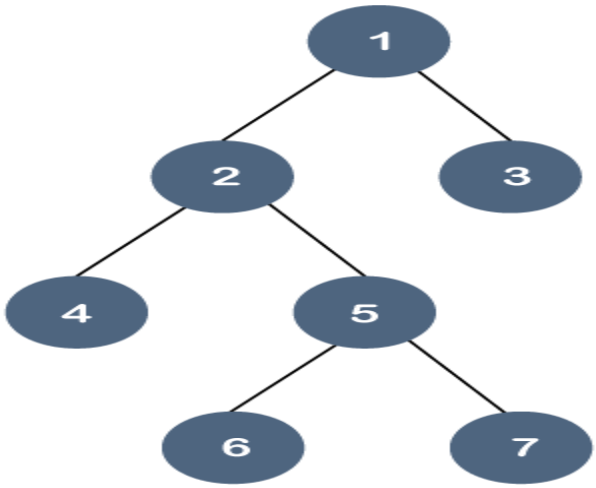
B tree is a self-balancing tree, and it is a m-way tree where m defines the order of the tree. **Btree** is a generalization of the Binary Search tree in which a node can have more than one key and more than two children depending upon the value of **m**. In the B tree, the data is specified in a sorted order having lower values on the left subtree and higher values in the right subtree.

Properties of B tree

The following are the properties of the B tree:

- In the B tree, all the leaf nodes must be at the same level, whereas, in the case of a binary tree, the leaf nodes can be at different levels.

Let's understand this property through an example.



In the above tree, all the leaf nodes are not at the same level, but they have the utmost two children. Therefore, we can say that the above tree is a binary tree but not a B tree.

- If the Btree has an order of m, then each node can have a maximum of **m** In the case of minimum children, the leaf nodes have zero children, the root node has two children, and the internal nodes have a ceiling of m/2.
- Each node can have maximum (m-1) keys. For example, if the value of m is 5 then the maximum value of keys is 4.
- The root node has minimum one key, whereas all the other nodes except the root node have (ceiling of m/2 minus - 1) minimum keys.
- If we perform insertion in the B tree, then the node is always inserted in the leaf node.

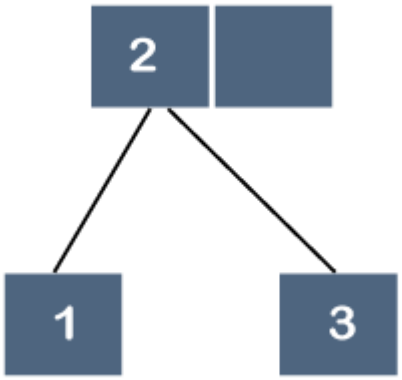
Suppose we want to create a B tree of order 3 by inserting values from 1 to 10.

Step 1: First, we create a node with 1 value as shown below:

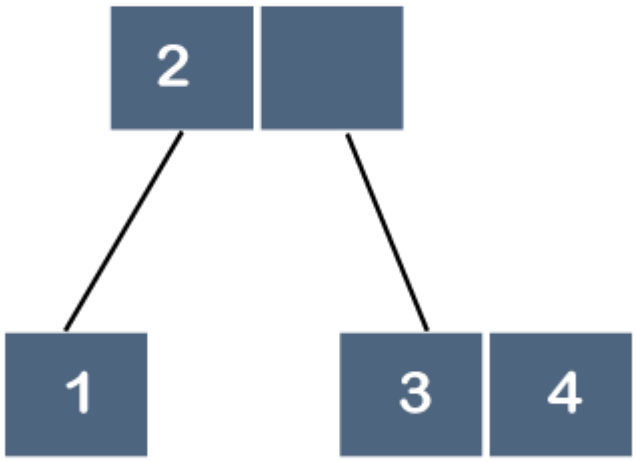
Step 2: The next element is 2, which comes after 1 as shown below:

Step 3: The next element is 3, and it is inserted after 2 as shown below:

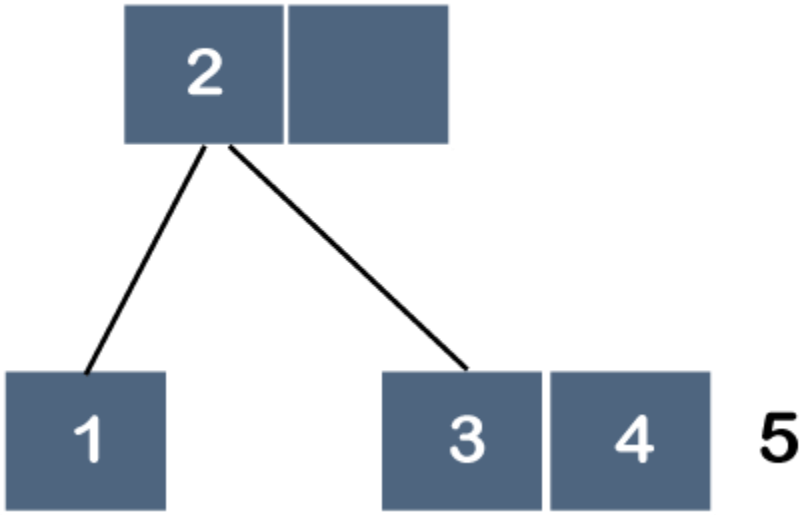
As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 2, so it moves to its parent. The node 2 does not have any parent, so it will become the root node as shown below:



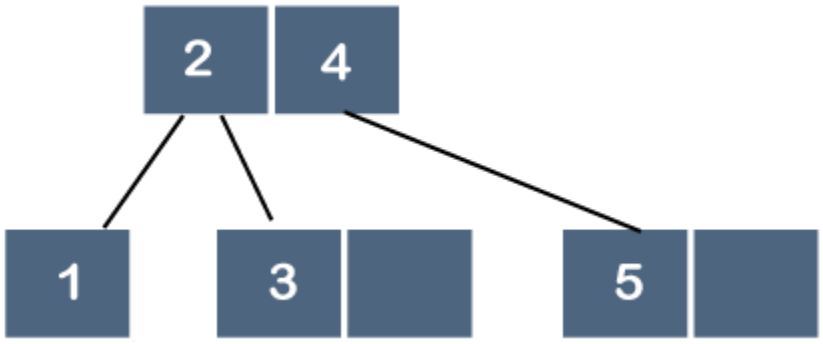
Step 4: The next element is 4. Since 4 is greater than 2 and 3, so it will be added after the 3 as shown below:



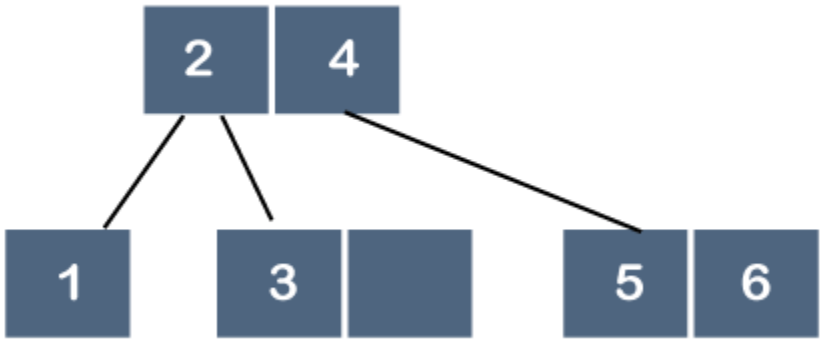
Step 5: The next element is 5. Since 5 is greater than 2, 3 and 4 so it will be added after 4 as shown below:



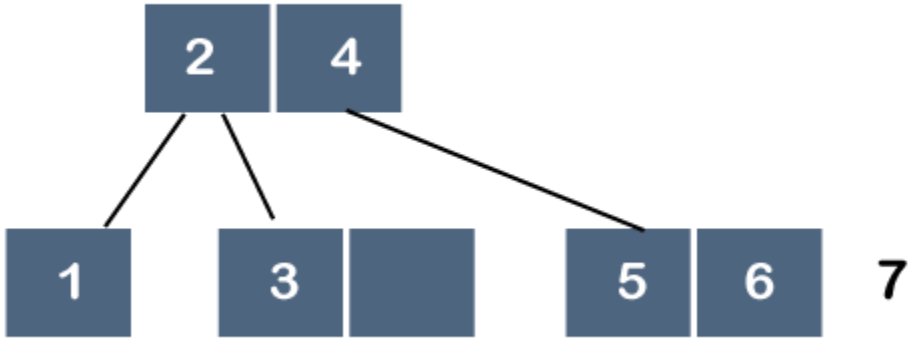
As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 4, so it moves to its parent. The parent is node 2; therefore, 4 will be added after 2 as shown below:



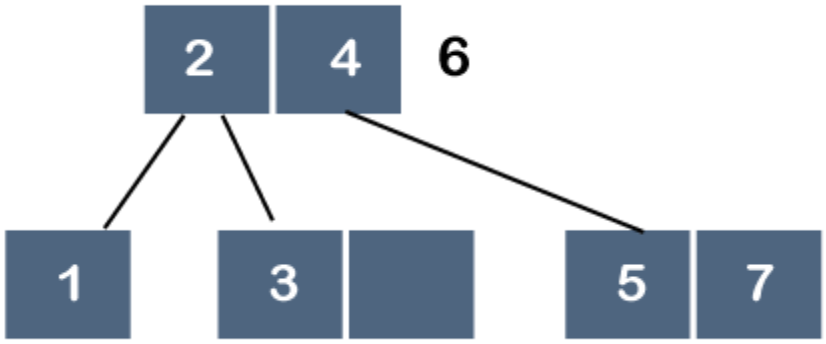
Step 6: The next element is 6. Since 6 is greater than 2, 4 and 5, so 6 will come after 5 as shown below:



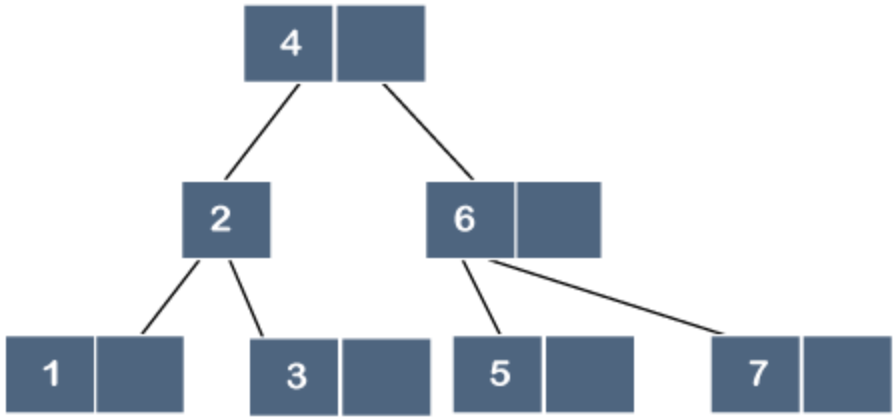
Step 7: The next element is 7. Since 7 is greater than 2, 4, 5 and 6, so 7 will come after 6 as shown below:



As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 6, so it moves to its parent as shown below:



But, 6 cannot be added after 4 because the node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 4, so it moves to its parent. As node 4 does not have any parent, node 4 will become a root node as shown below:



What is a B+ tree?

The B+ tree is also known as an advanced self-balanced tree because every path from the root of the tree to the leaf of the tree has the same length. Here, the same length means that all the leaf nodes occur at the same level. It will not happen that some of the leaf nodes occur at the third level and some of them at the second level.

A B+ tree index is considered a multi-level index, but the B+ tree structure is not similar to the multi-level index sequential files.

Why is the B+ tree used?

A B+ tree is used to store the records very efficiently by storing the records in an indexed manner using the B+ tree indexed structure. Due to the multi-level indexing, the data accessing becomes faster and easier.

B+ tree Node Structure

The node structure of the B+ tree contains pointers and key values shown in the below figure:

As we can observe in the above B+ tree node structure that it contains n-1 key values (k_1 to k_{n-1}) and n pointers (p_1 to p_n).

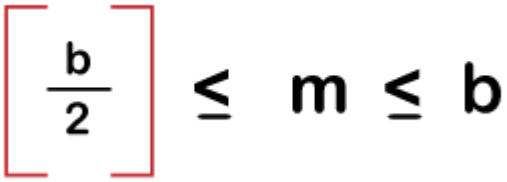
The search key values which are placed in the node are kept in sorted order. Thus, if $i<j$ then $k_i<k_j$.

Constraint on various types of nodes

Let 'b' be the order of the B+ tree.

Non-Leaf node

Let 'm' represents the number of children of a node, then the relation between the order of the tree and the number of children can be represented as:



Let k represents the search key values. The relation between the order of the tree and search key can be represented as:

As we know that the number of pointers is equal to the search key values plus 1, so mathematically, it can be written as:

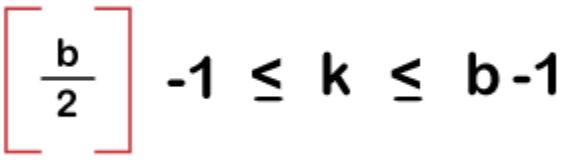
Number of Pointers (or children) = Number of Search keys + 1

Therefore, the maximum number of pointers would be 'b', and the minimum number of pointers would be the ceiling function of b/2.

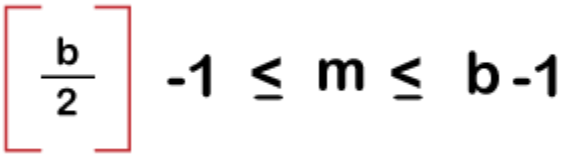
Leaf Node

A leaf node is a node that occurs at the last level of the B+ tree, and each leaf node uses only one pointer to connect with each other to provide the sequential access at the leaf level.

In leaf node, the maximum number of children is:



The maximum number of search keys is:



Root Node

The maximum number of children in the case of the root node is: b

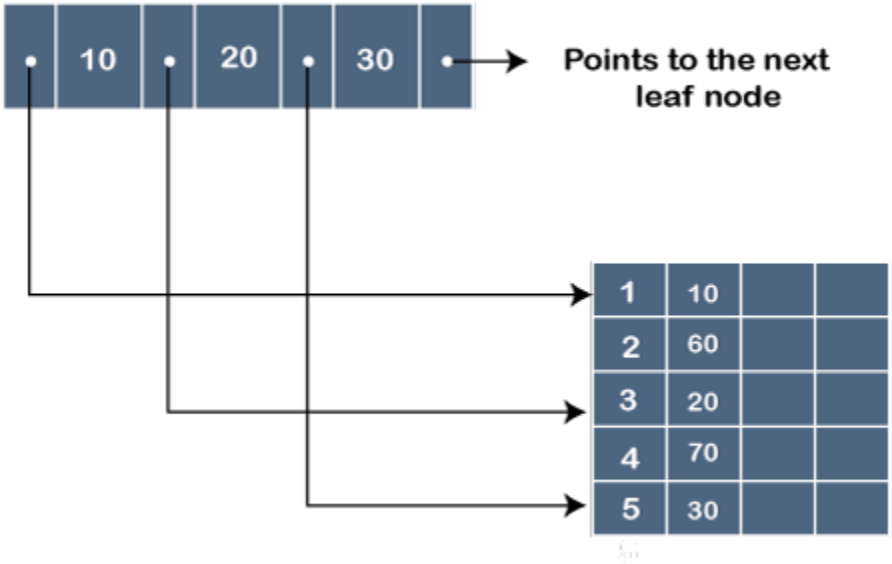
The minimum number of children is: 2

Special cases in B+ tree

Case 1: If the root node is the only node in the tree. In this case, the root node becomes the leaf node.

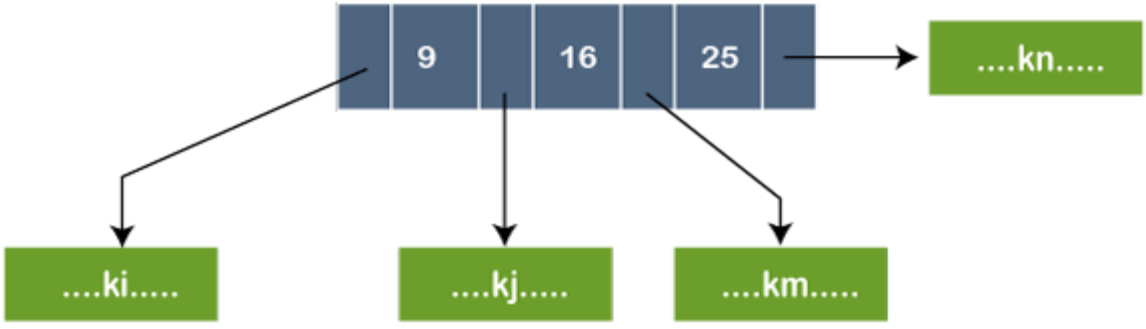
In this case, the maximum number of children is 1, i.e., the root node itself, whereas, the minimum number of children is b-1, which is the same as that of a leaf node.

Representation of a leaf node in B+ tree



In the above figure, '.' represents the pointer, whereas the 10, 20 and 30 are the key values. The pointer contains the address at which the key value is stored, as shown in the above figure.

Example of B+ tree



In the above figure, the node contains three key values, i.e., 9, 16, and 25. The pointer that appears before 9, contains the key values less than 9 represented by k_i. The pointer

that appears before 16, contains the key values greater than or equal to 9 but less than 16 represented by kj. The pointer that appears before 25, contains the key values greater than or equal to 16 but less than 25 represented by k_n.

Differences between B tree and B+ tree

The following are the differences between the B tree and B+ tree:

B tree	B+ tree
In the B tree, all the keys and records are stored in both internal as well as leaf nodes.	In the B+ tree, keys are the indexes stored in the internal nodes and records are stored in the leaf nodes.
In B tree, keys cannot be repeatedly stored, which means that there is no duplication of keys or records.	In the B+ tree, there can be redundancy in the occurrence of the keys. In this case, the records are stored in the leaf nodes, whereas the keys are stored in the internal nodes, so redundant keys can be present in the internal nodes.
In the Btree, leaf nodes are not linked to each other.	In B+ tree, the leaf nodes are linked to each other to provide the sequential access.
In Btree, searching is not very efficient because the records are either stored in leaf or internal nodes.	In B+ tree, searching is very efficient or quicker because all the records are stored in the leaf nodes.
Deletion of internal nodes is very slow and a time-consuming process as we need to consider the child of the deleted key also.	Deletion in B+ tree is very fast because all the records are stored in the leaf nodes so we do not have to consider the child of the node.

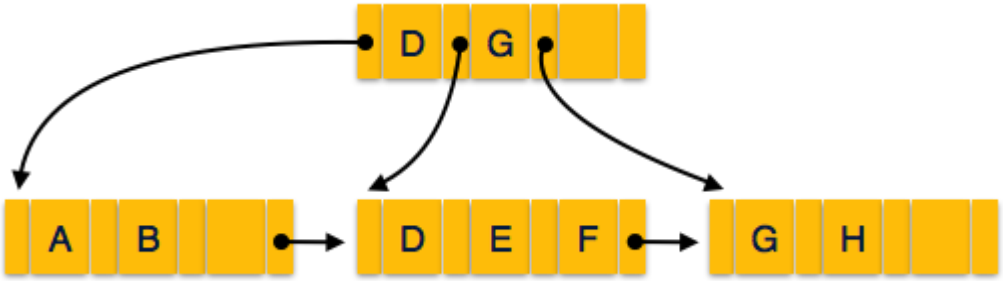
In Btree, sequential access is not possible.	In the B+ tree, all the leaf nodes are connected to each other through a pointer, so sequential access is possible.
In Btree, the more number of splitting operations are performed due to which height increases compared to width,	B+ tree has more width as compared to height.
In Btree, each node has atleast two branches and each node contains some records, so we do not need to traverse till the leaf nodes to get the data.	In B+ tree, internal nodes contain only pointers and leaf nodes contain records. All the leaf nodes are at the same level, so we need to traverse till the leaf nodes to get the data.
The root node contains atleast 2 to m children where m is the order of the tree.	The root node contains atleast 2 to m children where m is the order of the tree.

B^+ Tree

A B^+ tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B^+ tree denote actual data pointers. B^+ tree ensures that all leaf nodes remain at the same height, thus balanced. Additionally, the leaf nodes are linked using a link list; therefore, a B^+ tree can support random access as well as sequential access.

Structure of B^+ Tree

Every leaf node is at equal distance from the root node. A B^+ tree is of the order **n** where **n** is fixed for every B^+ tree.



Internal nodes –

- Internal (non-leaf) nodes contain at least [n/2] pointers, except the root node.
- At most, an internal node can contain **n** pointers.

Leaf nodes –

- Leaf nodes contain at least [n/2] record pointers and [n/2] key values.
- At most, a leaf node can contain **n** record pointers and **n** key values.
- Every leaf node contains one block pointer **P** to point to next leaf node and forms a linked list.

B^+ Tree Insertion

- B^+ trees are filled from bottom and each entry is done at the leaf node.
- If a leaf node overflows −
    - Split node into two parts.
    - Partition at **i = $\lfloor (m+1)/2 \rfloor$.**
    - First **i** entries are stored in one node.
    - Rest of the entries (i+1 onwards) are moved to a new node.
    - i^{th} key is duplicated at the parent of the leaf.
- If a non-leaf node overflows −
    - Split node into two parts.
    - Partition the node at **i = $\lceil (m+1)/2 \rceil$**.
    - Entries up to **i** are kept in one node.
    - Rest of the entries are moved to a new node.

B^+ Tree Deletion

- B^+ tree entries are deleted at the leaf nodes.
- The target entry is searched and deleted.
    - If it is an internal node, delete and replace with the entry from the left position.
- After deletion, underflow is tested,
    - If underflow occurs, distribute the entries from the nodes left to it.
- If distribution is not possible from left, then
    - Distribute from the nodes right to it.
- If distribution is not possible from left or from right, then
    - Merge the node with left and right to it.

ODBMS

The data model in which data is kept in the form of objects, which are instances of classes, is known as an object-oriented database management system, or ODBMS. The object-oriented data model is made up of these classes and objects.

A database management system (DBMS) that facilitates the modeling and generation of data as objects is called an object-oriented database management system (ODBMS), which is frequently abbreviated as ODBMS for object database management system. Inheritance of class attributes and methods by subclasses and their objects is also included, as is some sort of support for object classes.

Although the Object Data Management Group (ODMG) produced The Object Data Standard ODMG 3.0 in 2001, which outlines an object model and standards for defining and querying objects, there is no commonly accepted definition of what an OODBMS is. Since then, the group has broken up.

Nowadays, a well-liked substitute for the object database is Not Only SQL (NoSQL) document database systems. NoSQL document databases offer key-based access to semi-structured data as documents, generally using JavaScript Object Notation, even if they lack all the features of a full ODBMS (JSON).

Object-Oriented Data Model Elements

The three main building blocks of the OODBMS are object structure, object classes, and object identity. The following explains them.

Object Structure: An object's structure refers to the components that make up the object. An attribute is a term used to describe certain characteristics of an item. A real-world entity with certain qualities that makes up the object structure is hence referred to as an object. Also, an object contains the data code in a solitary piece, which in turn creates data abstraction by shielding the user from the implementation specifics.

1. Messages - A message operates as a communication channel or as an interface between an entity and the outside world. There are two sorts of messages:
2. Read-only message: The invoking message is referred to as a read-only message if the called method does not alter the value of a variable.
3. Update message: The invoking message is referred to as an update message if the invoked method modifies the value of a variable.
4. Methods: A method is a chunk of code that is executed when a message is given. Every time a method is used, a value is returned as output. There are two categories of methods:
5. Read-only method: A method is referred to as a read-only method when it has no effect on the value of a variable.
6. Update-method: An update method is one that modifies a variable's value in some way.
7. Variables are used to store an object's data. The variables' data allows the objects to be distinguished from one another.

Object Classes: An instance of a class is an object, which is a real-world item. In order to create objects that differ in the data they hold but have the same class definition, a class must first be defined. Messages and variables recorded in the objects themselves relate to the objects in turn.

```
1. class JOB
2.   { //variables
3.       char name;
4.       string address;
5.       int id;
6.       int salary;
7.      //Messages
8.       char get_name();
```

```
9.      string get_address();
10.     int annual_salary();
11.  };
```

As we can see in the sample above, the class CLERK is where the object variables and messages are stored.

Although there may be several classes with comparable methods, variables, and messages in a database, an OODBMS also extensively allows inheritance. As a result, the idea of the class hierarchy is still used to illustrate the commonalities between different classes.

An object-oriented data model also supports the idea of encapsulation, which is data or information concealing. In addition to the built-in data types like char, int, and float, this data architecture also offers the ability for abstract data types. ADTs are user-defined data types that can carry methods in addition to the values they contain.

Hence, ODBMS offers a variety of built-in and user-defined features to its customers. It combines the attributes of an object-oriented data model with those of a database management system, and it supports the notions of programming paradigms like classes and objects in addition to those of encapsulation, inheritance, and user-defined ADTs (abstract data types).

Features of ODBMS

Malcolm Atkinson and colleagues defined an OODBMS in their important work, The Object-Oriented Database Manifesto, as follows:

An object-oriented database system must meet two requirements: it must be a database management system (DBMS) and it must be an object-oriented system, meaning that it must, to the greatest degree feasible, be compatible with the current crop of object-oriented programming languages.

Persistence, secondary storage management, concurrency, recovery, and an ad hoc query capability are the five qualities that correspond to the first criteria.

The second one translates into eight characteristics: extensibility, computational completeness, overriding paired with late binding, inheritance, types or classes, complex objects, object identity, and encapsulation.

RDBMS Vs ODBMS

The type of DBMS that is now used the most frequently is a relational database management system (RDBMS). Most IT workers have a solid understanding of the relational abstraction of rows and columns accessible via Structured Query Language (SQL).

Yet, object database systems may be more effective in managing and storing complicated data relationships. Accessing data with several relationships spread across various tables in an RDBMS might be more challenging for applications than accessing the same data as an object in an ODBMS.

In addition, a lot of programmers employ object-oriented programming (OOP) languages to create applications, including Java, C++, Python, and others. Conversions between complicated objects and rows from various relational database tables are necessary when using an RDBMS to store and retrieve objects. Tools for object-relational mapping (ORM) can make this process simpler; nonetheless keep the mapping overhead in place.

Several RDBMS companies now provide object-relational database management systems as part of their product lines (ORDBMS). Of fact, adding some object-oriented ideas to relational databases does not give users access to all of an ODBMS's features.

Distributed Database System in DBMS

A distributed database is essentially a database that is dispersed across numerous sites, i.e., on various computers or over a network of computers, and is not restricted to a single system. A distributed database system is spread across several locations with distinct physical components. This can be necessary when different people from all over the world need to access a certain database. It must be handled such that, to users, it seems to be a single database.

Types:

1. **Homogeneous Database:** A homogeneous database stores data uniformly across all locations. All sites utilize the same operating system, database management system, and data structures. They are therefore simple to handle.

2. **Heterogeneous Database:** With a heterogeneous distributed database, many locations may employ various software and schema, which may cause issues with queries and transactions. Moreover, one site could not be even aware of the existence of the other sites. Various operating systems and database applications may be used by various machines. They could even employ separate database data models. Translations are therefore necessary for communication across various sites.

Data may be stored on several places in two ways using distributed data storage:

1. **Replication -** With this strategy, every aspect of the connection is redundantly kept at two or more locations. It is a completely redundant database if the entire database is accessible from every location. Systems preserve copies of the data as a result of replication. This has advantages since it makes more data accessible at many locations.

Moreover, query requests can now be handled in parallel. But, there are some drawbacks as well. Data must be updated often. All changes performed at one site must be documented at every site where that relation is stored in order to avoid inconsistent results. There is a tone of overhead here. Moreover, since concurrent access must now be monitored across several sites, concurrency management becomes far more complicated.

2. **Fragmentation -** In this method, the relationships are broken up into smaller pieces and each fragment is kept in the many locations where it is needed. To ensure there is no data loss, the pieces must be created in a way that allows for the reconstruction of the original relation. As fragmentation doesn't result in duplicate data, consistency is not a concern.

Uses for distributed databases

- The corporate management information system makes use of it.
- Multimedia apps utilize it.
- Used in hotel chains, military command systems, etc.
- The production control system also makes use of it

Characteristics of distributed databases

Distributed databases are logically connected to one another when they are part of a collection, and they frequently form a single logical database. Data is physically stored across several sites and is separately handled in distributed databases. Each site's processors are connected to one another through a network, but they are not set up for multiprocessing.

A widespread misunderstanding is that a distributed database is equivalent to a loosely coupled file system. It's considerably more difficult than that in reality. Although distributed databases use transaction processing, they are not the same as systems that use it.

distributed databases have the following characteristics:

- Place unrelated
- Spread-out query processing
- The administration of distributed transactions
- Independent of hardware
- Network independent of operating systems
- Transparency of transactions
- DBMS unrelated

Architecture for a distributed database

Both homogeneous and heterogeneous distributed databases exist.

All of the physical sites in a homogeneous distributed database system use the same operating system and database software, as well as the same underlying hardware. It can be significantly simpler to build and administer homogenous distributed database systems since they seem to the user as a single system. The data structures at each site must either be the same or compatible for a distributed database system to be considered homogeneous. Also, the database program utilized at each site must be compatible or same.

The hardware, operating systems, or database software at each site may vary in a heterogeneous distributed database. Although separate sites may employ various technologies and schemas, a variation in schema might make query and transaction processing challenging.

Various nodes could have dissimilar hardware, software, and data structures, or they might be situated in incompatible places. Users may be able to access data stored at a different place but not upload or modify it. Because heterogeneous distributed databases are sometimes challenging to use, many organizations find them to be economically unviable.

Distributed databases' benefits

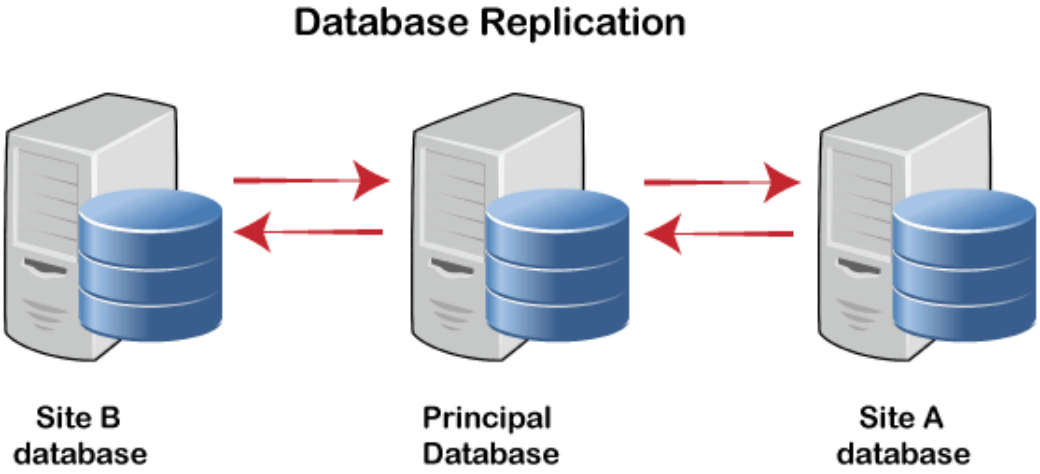
Using distributed databases has a lot of benefits.

- As distributed databases provide modular development, systems may be enlarged by putting new computers and local data in a new location and seamlessly connecting them to the distributed system.
- With centralized databases, failures result in a total shutdown of the system. Distributed database systems, however, continue to operate with lower performance when a component fails until the issue is resolved.
- If the data is near to where it is most often utilized, administrators can reduce transmission costs for distributed database systems. Centralized systems are unable to accommodate this<

Types of Distributed Database

- Data instances are created in various areas of the database using replicated data. Distributed databases may access identical data locally by utilizing duplicated data, which reduces bandwidth. Read-only and writable data are the two types of replicated data that may be distinguished.

- Only the initial instance of replicated data can be changed in read-only versions; all subsequent corporate data replications are then updated. Data that is writable can be modified, but only the initial occurrence is affected.



- Primary keys that point to a single database record are used to identify horizontally fragmented data. Horizontal fragmentation is typically used when business locations only want access to the database for their own branch.
- Using primary keys that are duplicates of each other and accessible to each branch of the database is how vertically fragmented data is organized. When a company's branch and central location deal with the same accounts differently, vertically fragmented data is used.
- Data that has been edited or modified for decision support databases is referred to as reorganised data. When two distinct systems are managing transactions and decision support, reorganised data is generally utilised. When there are numerous requests, online transaction processing must be reconfigured, and decision support systems might be challenging to manage.
- In order to accommodate various departments and circumstances, separate schema data separates the database and the software used to access it. Often, there is overlap between many databases and separate schema data

Distributed database examples

- Apache Ignite, Apache Cassandra, Apache HBase, Couchbase Server, Amazon SimpleDB, Clusterpoint, and FoundationDB are just a few examples of the numerous distributed databases available.
- Large data sets may be stored and processed with Apache Ignite across node clusters. GridGain Systems released Ignite as open source in 2014, and it was later approved into the Apache Incubator program. RAM serves as the database's primary processing and storage layer in Apache Ignite.

- Apache Cassandra has its own query language, Cassandra Query Language, and it supports clusters that span several locations (CQL). Replication tactics in Cassandra may also be customized.
- Apache HBase offers a fault-tolerant mechanism to store huge amounts of sparse data on top of the Hadoop Distributed File System. Moreover, it offers per-column Bloom filters, in-memory execution, and compression. Although Apache Phoenix offers a SQL layer for HBase, HBase is not meant to replace SQL databases.
- An interactive application that serves several concurrent users by producing, storing, retrieving, aggregating, altering, and displaying data is best served by Couchbase Server, a NoSQL software package. Scalable key value and JSON document access is provided by Couchbase Server to satisfy these various application demands.
- Along with Amazon S3 and Amazon Elastic Compute Cloud, Amazon SimpleDB is utilised as a web service. Developers may request and store data with Amazon SimpleDB with a minimum of database maintenance and administrative work.
- Relational database designs' complexity, scalability problems, and performance restrictions are all eliminated with Clusterpoint. Open APIs are used to handle data in the XLM or JSON formats. Clusterpoint does not have the scalability or performance difficulties that other relational database systems experience since it is a schema-free document database.