UNIT 1

UNIT - I Database System Applications: A Historical Perspective, File Systems versus a DBMS, the Data Model, Levels of Abstraction in a DBMS, Data Independence, Structure of a DBMS Introduction to Database Design: Database Design and ER Diagrams, Entities, Attributes, and Entity Sets, Relationships and Relationship Sets, Additional Features of the ER Model, Conceptual Design With the ER Model

- Applications of Database System

The applications of database systems are wide and it never ends up. Some of the application areas include

1. University 2. Banking 3. Hospital 4. Telecommunication 5. Finance
6. Sales and marketing

- History of database

The database system has a long history over the period the technology for database storage and access method has been changed.

1. During 1950's, during this period, magnetic tapes are used as storage media. Data will be stored and processed on the magnetic tapes by using sequential access. Therefore, processing speed is less during 1960's to 1970's.
2. During this period, the usage of hard disk has changed the scenario of data storage and data processing. The hard disk will provide direct access of the data. Therefore, process will become faster.
3. During 1970's, E.F. Codd introduced the relational model and therefore many relational DB has been started.
4. During 1980's, during this period, many relational DB such as oracle, SQL server, IBM DML has been introduced in the market and many researchers start working on disturbed databases during 1990's many database vendors provided many distributed databases into market and the usage of SQL has provided a convenient environment for the user to work with database system.

- File System vs Database system

In early days, before database systems were introduced, users had to store their data in files and retrieve the data from the files by writing different application programs this technique is known as File Processing System (FPS).

File Processing System is suitable with the collection of files is less in number and data is limited. When the file size increases, it becomes difficult to maintain such data with FPS.

* **Drawbacks of File Processing System**

a. Data Redundancy b. Data inconsistency c. Data Integrity
d. Data Isolation e. Data Security f. Difficulty in accessing the data

a. Data Redundancy:

It means duplication of the data. It leads to the wastage of storage space. This happens because no

validation methods available in File Processing System.

b. Data Inconsistency:

Data Redundancy leads to a problem known as Data Inconsistency i.e. multiple copies of the same data may no longer agree with each other.

c. Data Integrity:

It refers to correctness. Integrity problem will arise due to the lack of integrity checks such as student age should not be less than 18 etc.

d. Data Isolation:

In File System, data is distributed in different locations and to retrieve the data from these isolated files, large application programs need to be written.

e. Data Security:

In File System, the files can be password protected sometimes we want to give the access to few records from a file then security becomes difficult.

f. Difficulty in accessing the data:

In File System, efficient data access methods are not available hence accessing the data is difficult.

- View of Data

Database is a collection of large volumes of data the user does not always require the entire data from the

database. Therefore, it is the responsibility of the database to provide the required data to the user.

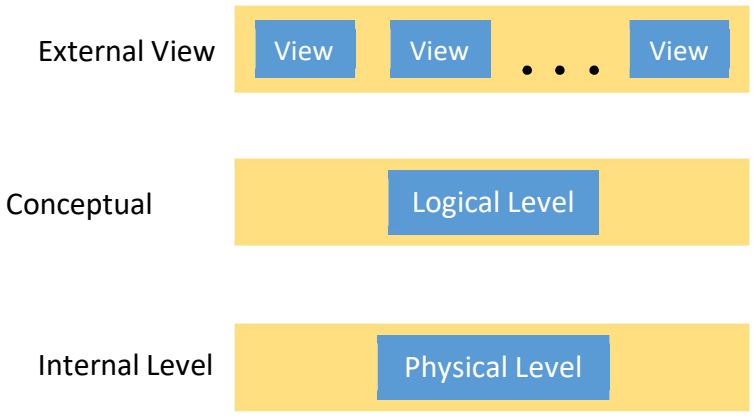
- Data abstraction

Data abstraction refers to the way of representing the essential features and hiding background details or complexities from the user.

- Need for the data abstraction

Data abstraction is necessary because user is not computer trained or expert. To make user job simpler different levels of data abstraction are provided so that user will feel convenient to work with the database.

- Levels of abstraction



There are 3 levels of data abstraction

1. Physical Level 2. Logical Level 3. View Level

1. **Physical Level**
   a. It is also known as internal level or lower level.
   b. It describes about how the data is actually stored in the database (Implementation or storage structure).
2. **Logical Level**
   a. It is also known as conceptual level.
   b. It describes what data is actually stored in the database and relationship among the data.
   c. Logical design is taken by DBA (Data Base Administrator).
3. **View Level**
   a. It is also known as external level.
   b. It describes the part of entire database in the form of different views by different users.

- Instance

A database instance is a set of memory structure and background processes that access a set of database files. The process can be shared by all users. The memory structure that are used to store most queried data from database. This helps up to improve database performance by decreasing the amount of I/O performed against data file.

- Schema/Scheme

The overall design of the database is known as the database schema. According to the levels of data abstractions, there are 3 types of schemas available.

1. Physical Scheme 2. Logical Scheme 3. Sub Scheme

1. **Physical Scheme:** It describes the structure of data at physical level.
2. **Logical Scheme:** It describes the structure of data at logical level.
3. **Sub Scheme:** It describes different values of the users interacting with the database.

- Data independence

It is defined as the ability to modify data at one level without effecting at the next level. Data independence is of 2 types

1. Physical data independence 2. Logical data independence

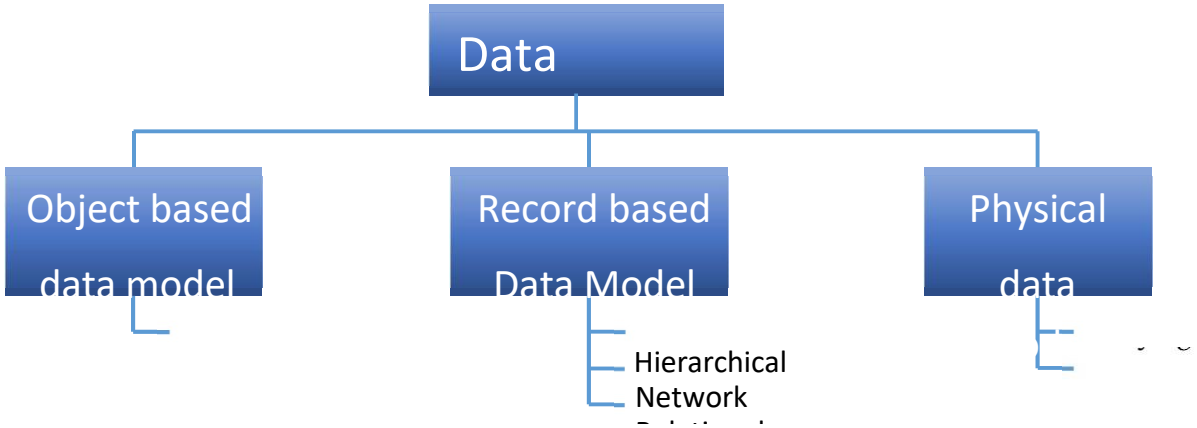
1. **Physical Data Independence:**
   The ability to modify the physical structure at the physical level without effecting the next level (Logical level) is known as Physical Data Independence.
2. **Logical Data Independence:**
   The ability to modify the data at the logical level without effecting the next level (View level/External level) is known as Logical Data Independence.
   Note: Logical Data Independence is more difficult to implement than Physical Data Independence.

- Data Models

Data model is the way to represent the data within the database. Data model is a collection of conceptual tools for describing:

1. For describing data
2. The relationship among data
3. Data semantic
4. The consistency constraints

Different data models are available. The classification of data model is shown below



Data model is mainly classified into 3 categories:

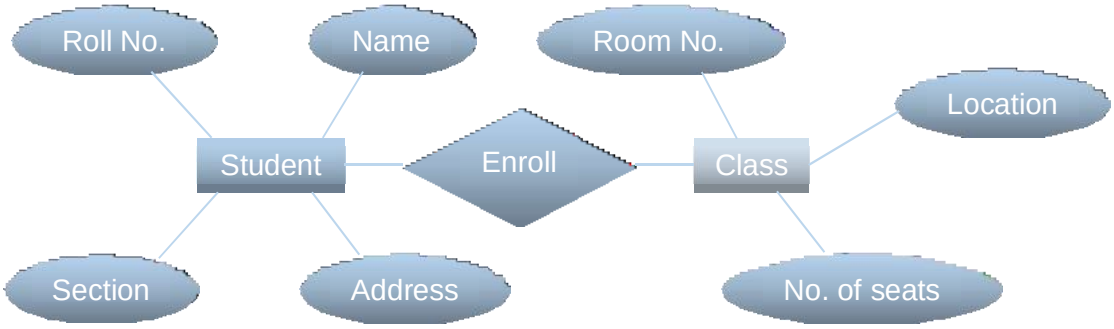
1. Object based data model 2. Record based data model 3. Physical data model

1. **Object based data model:**
   The object-based data model deals with the real-world object and relation among the objects. One of the popular models to represent object-based model in E-R model.

   E-R Model (Entity-Relationship):
   The overall design of the database is represented graphically using entity-relationship diagram. The E-R model shows entities, relations among entities in a diagrammatic fashion.
   Ex: Entity: Student Entity: Class

Attributes: Roll no.
Name
Address
Section

Attributes: Room no.
No. of seats
Location



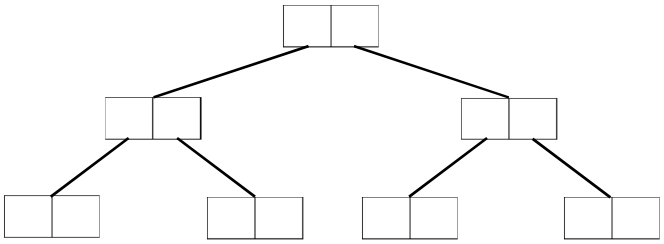
2. Record based data model:

The record-based data model stores the data in the form of fixed format record where each record maybe having fixed length. Record-based data model contains 3 models

a. Hierarchical model b. Network model c. Relational model

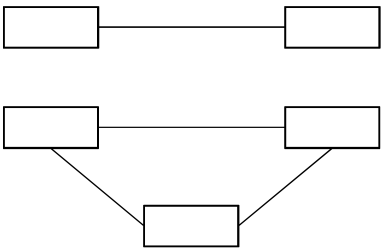
a. Hierarchical Model:

In this model, the data and relationship among data is represented by using records and link or pointer. This model is generally in the form of tree like structure.



b. Network Model:

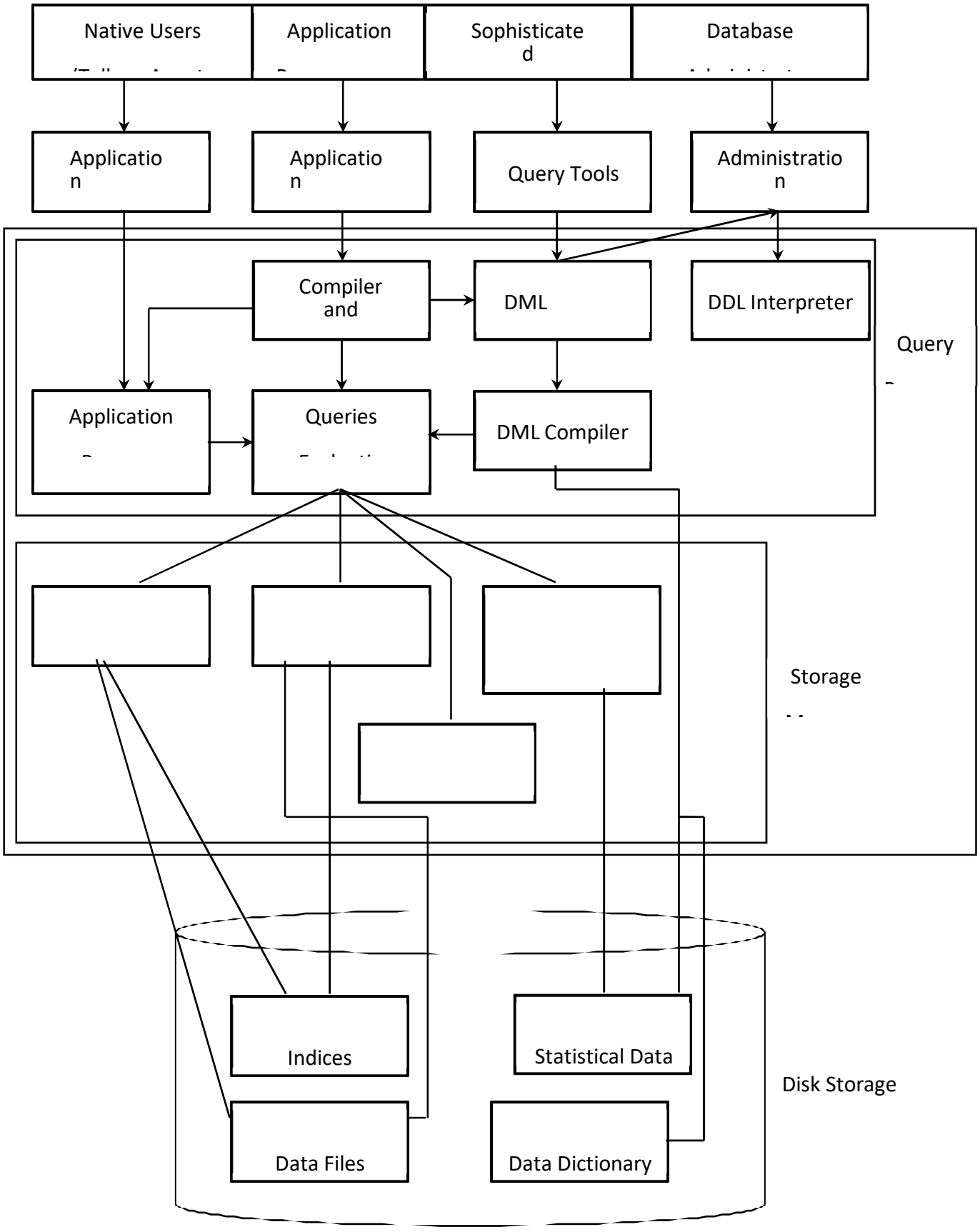
In this model, data and relation among data is represented using records and pointers. This model is generally represented in graph like structure.



c. Relational Model:

i. It is the most popular data model used nowadays.
ii. It was developed in year 1970 by EF Codd.
iii. The main constraint is to represent the relational model is in the form of relation (Table).
iv. A relation is a combination of rows and columns.
v. Each relation represents a relational schema and relational instance.

- Database system structure:



- Database User & Database Administrator:

The people who work with the database is categorised into 2 types:

1. Database User 2. Database Administrator

1. Database User

The database users are categorised into 4 types according to the way they are expected to interact with

database:

a. Native User (Unsophisticated) b. Application Programmers

c. Sophisticated User (Analyst) d. Specialized User

a. Native User (Unsophisticated):

* Native user also known as unsophisticated user or unskilled user.
* They interact with the database by using an API (Application Program Interface).
  Ex: The users of ATM i.e. Automatic Telling Machine are categories as native user, teller, agents.

b. Application Programmers:

Application programmers are the skilled users who write certain application programs to interact with the database. They use RAD (Rapid Application Development) tools to write the application programs. They use the interface such as forms reports.

c. Sophisticated User:

* Sophisticated users also known as analyst.
* They interact with the database by using DML queries.
* These DML queries will be compiled using DML compiler and evaluated using query evaluation engine.

d. Specialized User:

The specialized user are the sophisticated users who write some complex application programs such as Computer Aided Design (CAD), Artificial Intelligence (AI), expert system & some graphics-based application.

2. Database Administrator (DBA):

DBA is a person who has the centralised control over the entire database.

* **Functions of DBA**

a. Scheme definition:

The DBA is responsible for defining the schema (overall design) of the database.

b. Storage structure of access method definition:

The DBA is responsible to define the storage structure (logical design) and access method (Retrieval Technology).

c. Physical organisation modification:

The DBA is responsible for the modification of the physical organisation of the database.

d. Granting authorization for data access:

The DBA will provide the access permissions to the user so that only the authorised users will access the database.

e. Regular Maintenance:

i. Taking backup:

The DBA will take the backup of the database at regular intervals of time to be used for recovery purpose.

ii. Monitoring the jobs:

The DBA will monitor the running jobs in order to maintain the performance of the system.

iii. Monitoring the disk space:

The DBA will monitor the file space allocation so that the new jobs will get the needed space in the disk.

- Database Architecture:

The functional components of database architecture are categorised into 2 types:

1. **Query processor** 2. **Storage manager**

1. **Query processor:**

Query processor is a major component of database architecture. It includes the following components:

a. **DML Compiler:**

The DML compiler is used to compile the DML queries submitted by the user and generate low level instruction understood by query evaluation engine.

b. **DDL Interpreter:**

The DDL interpreter will execute the DDL statements given by the user and store the result in a special file called Data Dictionary.

c. **Query evaluation engine:**

The query evaluation will execute or evaluate low level instructions by DML compiler.

2. **Storage manager:**

The storage manager is a program module that acts as an interface between the query processor and low-level data stored in the disk storage. It includes the following components:

a. **Buffer manager:**

The buffer manager is responsible for allocating temporary storage for the files.

b. **File manager:**

The file manager will take care of the files being stored in the database.

c. **Authorization & Integrity manager:**

This component is responsible for giving the authorization DAP (Data Access Permission) and maintaining integrity of the database.

d. **Transaction Manager:**

It is responsible for managing the transaction within the database system. A transaction is a logical unit of work done by the user.

Ex: Credit & debit operation in bank transaction.

The storage will maintain several data structures as part of physical storage implementation.

a. **Indices (indexes):** for faster retrieval of data.
b. **Data files:** It is a collection of data stored in the files.
c. **Data Dictionary:** It is a container for meta data.
d. **Statistical data:** It maintain the statistical information of database users.

- Database Design

The database design process consistence of the following steps:

1. **Requirement Analysis (Data Gathering)**
2. **Conceptual Design (ER Model)**
3. **Logical Design (Relational Model)**

4. **Schema Refinement (Normalization Techniques)**
5. **Physical Database Design (index, clusters etc)**
6. **Database Tuning**
7. **Security Design (Authorization)**

- ER Model beyond ER Design

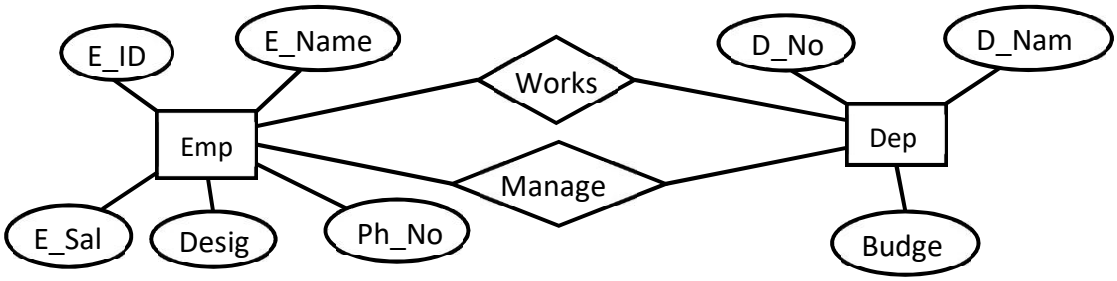
The ER model consistence of various entity objects, attributes of entity properties and relationship among the entities are represented in a diagrammatic fashion is known as entity relationship model.

Ex: The company database wants to maintain the following information about their emp and department. The emp (E_ID, E_Name, E_Sal, Desig, Ph_No), Dept (D_No, D_Name, Budget). Identify the key attributes and the relationship among the entity is as follows

There are 2 conditions

1. Emp works in department
2. Each department is managed by the employee

Draw a neat sketch diagram for the above information



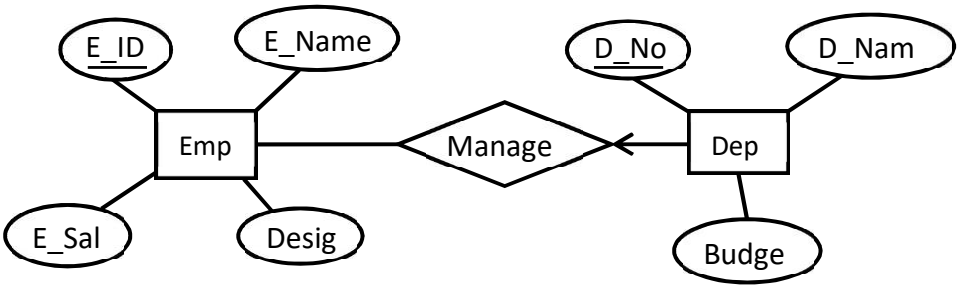
- Additional features of ER model

The ER model consist of the following features

1. Key constrains 2. Participation constrains 3. Weak entities 4. Class hierarchy
5. Aggregation 6. Key constraint for ternary relationship

1. Key Constraints:

Consider the following ER – diagram given below



In the above ER diagram, there is a restriction that each department is managed by an employee (Manager). This restriction is an example for the key constraints. The key constraints are represented with an arrow in the diagram.

2. Participation constraints:

The participation is of 2 types

a. Total participation

b. Partial participation

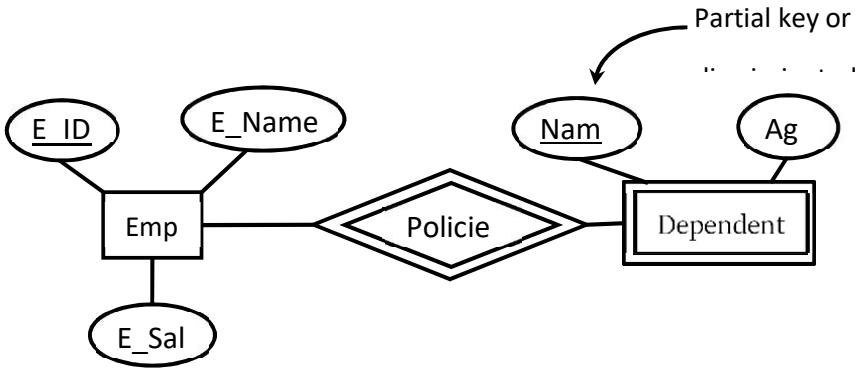
Consider the ER diagram given above, in the ER diagram, the dept entity is totally participating with the managers relationship. This participation is known as total participation.

If the participation is not total, then it is said to be partial. In the above ER diagram, the emp entity is partially participating with the manages relationship. Hence, it is called as partial participation.

3. Weak Entities:

A weak entity is an entity that does not contain a primary key. It is represented with double rectangle box

Consider the given ER diagram

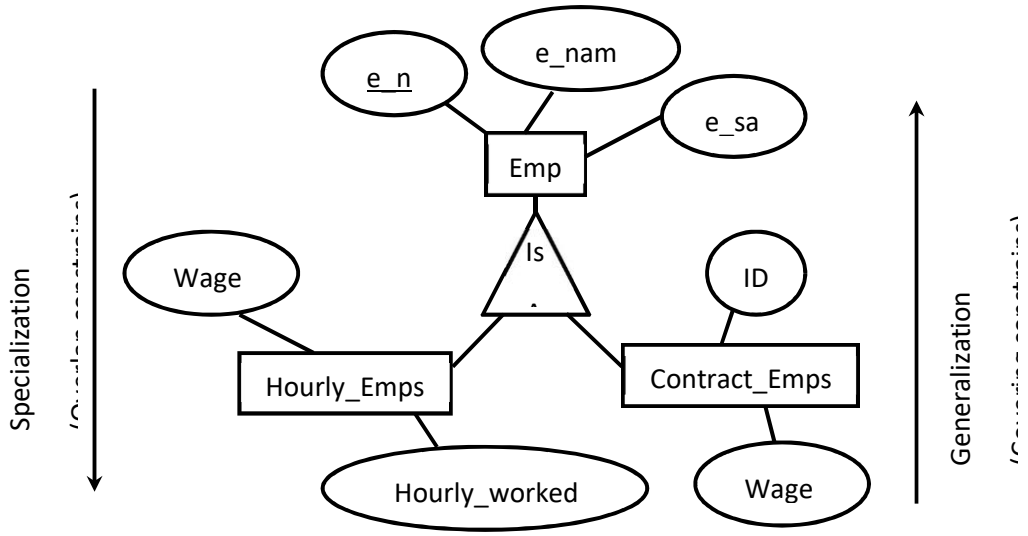


In the given example, dependent is a weak entity.

4. Class Hierarchy:

Sometimes it is common to represent entity into subclass using 'IS A' relationship. This concept is known as a class hierarchy. The class hierarchy represents the inheritance concept where a super class may have some sub classes. The class hierarchy is represented in 2 ways

a. Generalization b. Specialization



a. Generalization:

Generalization is the process of finding some common properties of two sub classes having a super class entity. In the above example, hourly emp, contract emp are generalized in emp.

b. Specialization:

The process of sub-dividing a super class entity into sub class entity is known as specialization. In the above example, the super class entity emp is having sub class entities – hourly employee and contract employee.

There are 2 constrains w.r.t generalization & specialization

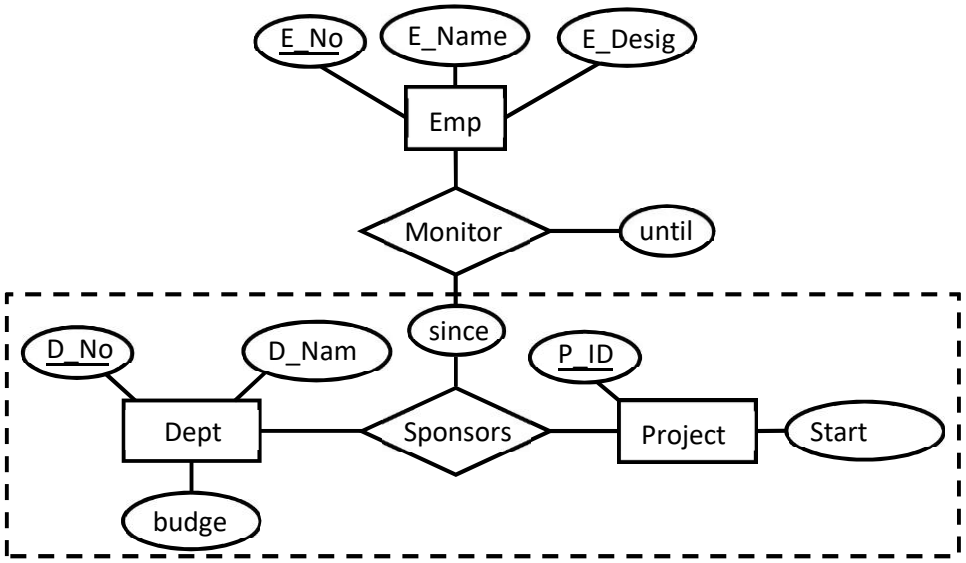
i. Overlap Constraint:

The overlap constraint determines whether two subclass entities are allowed to have common attributes of superclass.

ii. Covering Constraint:

Covering constraint determines whether the subclass entities include all the attributes of super class.

5. Aggregation



In ER diagram, we represent relationship as an association among 2 entities. Sometimes we want to represent relationship among relationships. This will be done using a concept known as Aggregation. In the above example, the relationship set sponsors is associated with the relationship monitors.

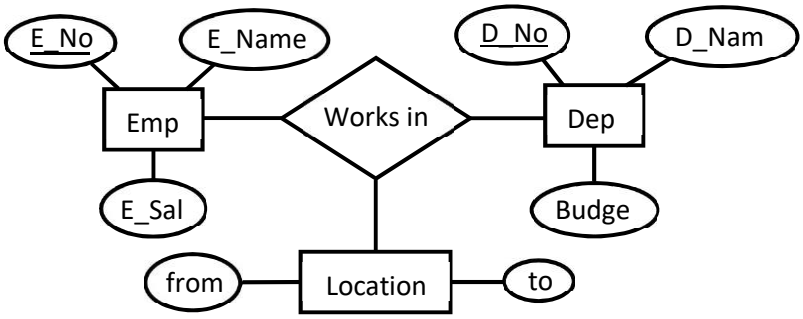
The aggregation is represented by considering the relationship set sponsored among 2 entities,

departments, projects as an entity set. It is shown with a dotted box in the diagram.

- Features of Aggregation

Aggregation is used to express relationship among relationship

6. Key constraints for ternary relationship



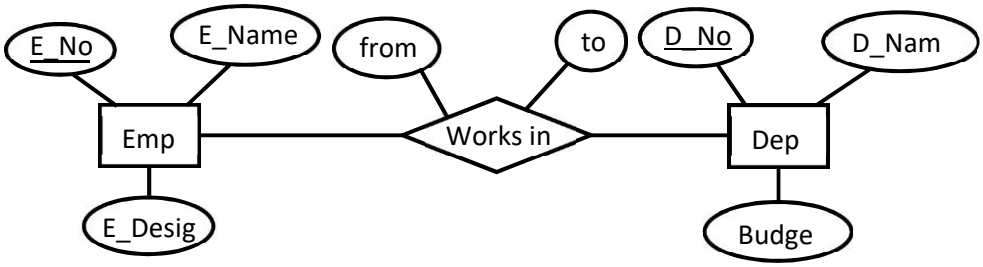
In the given ER diagram, an employee works in a department and in a single location. This restriction for ternary relationship is represented with the key constraints using an arrow from employees to work in relationship.

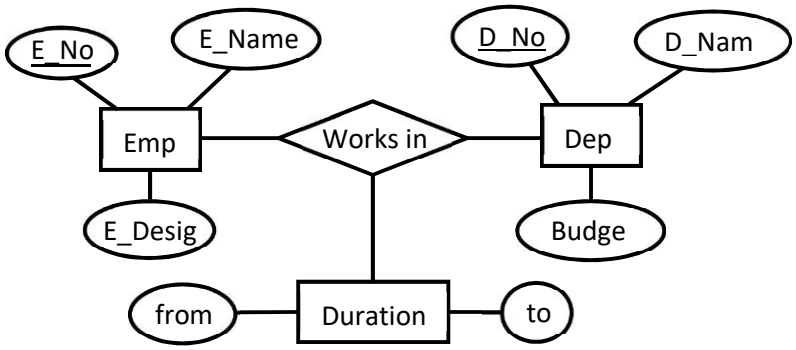
- Conceptual design with the ER Model

Conceptual design is the process of defining a high-level description of the data using ER model. There are different design issues while designing conceptual design with ER model.

1. Entity vs Attribute

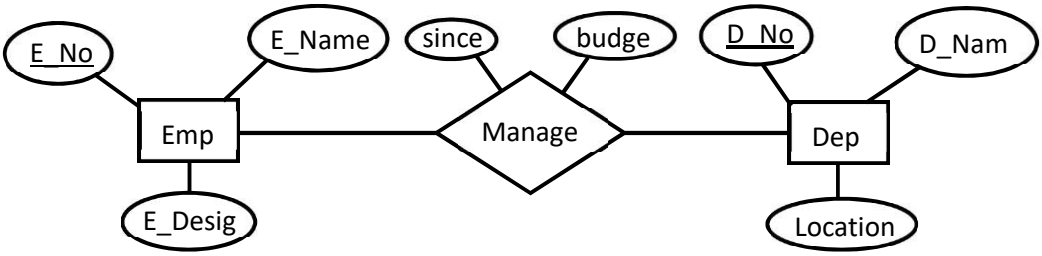
Sometimes an attribute of an entity set can be better represented as entities. Consider the ER diagram.

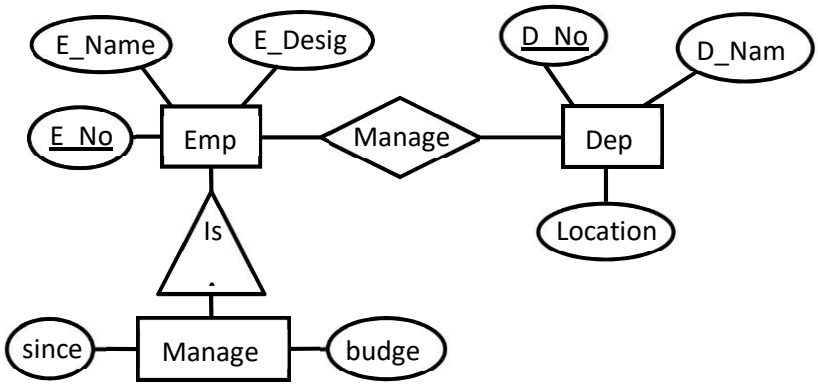




2. Entity vs Relationship

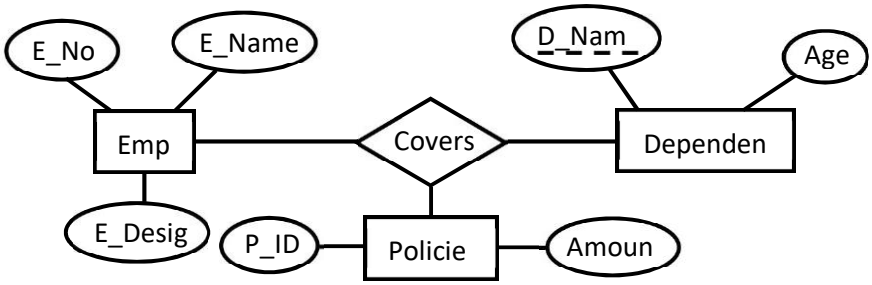
Sometimes an object will be better expressed as an entity rather than a relation.

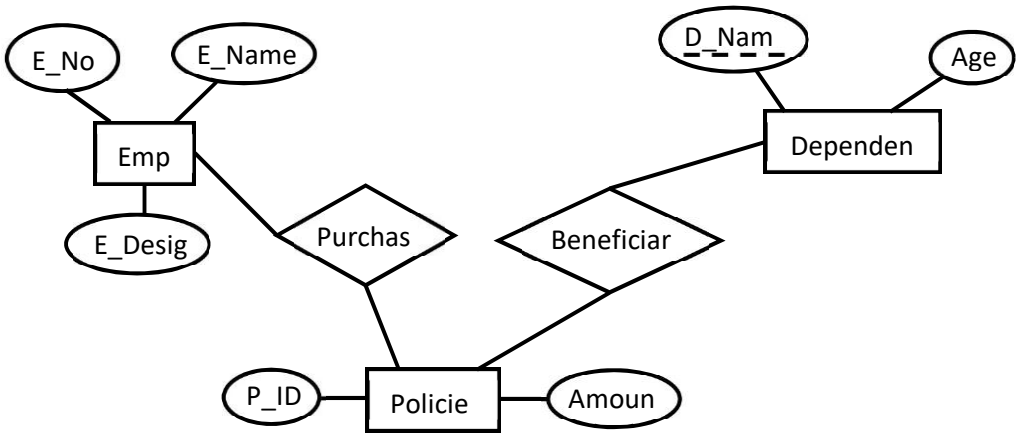




3. Binary vs Ternary

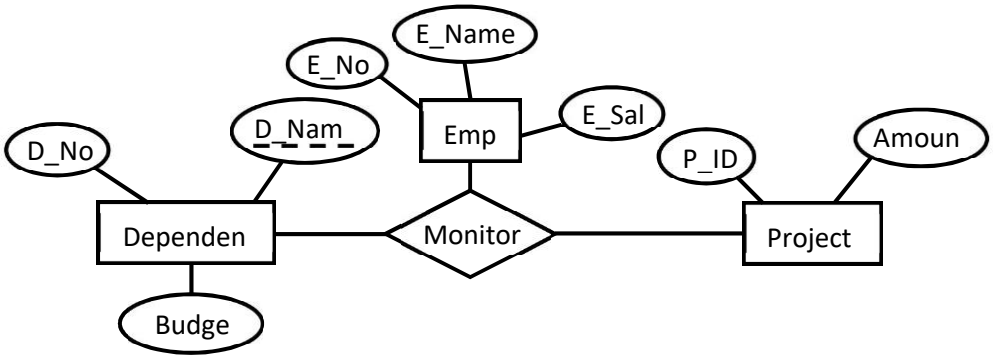
Sometimes a non - binary relationship can be expressed using distinct binary relations. Consider the givenER diagram policies with ternary relation.





4. Aggregation vs Ternary

Sometimes an ER diagram with aggregation can be best expressed as ternary relation.



Entity

Any thing that has an independent existence and about which we collect data. It is also known as entity type.

In ER modeling, notation for entity is given below.

Entity instance

Entity instance is a particular member of the entity type.

Example for entity instance : A particular employee

Regular Entity

An entity which has its own key attribute is a regular entity.

Example for regular entity : Employee.

Weak entity

An entity which depends on other entity for its existence and doesn't have any key attribute of its own is a weak

entity.

Example for a weak entity : In a parent/child relationship, a parent is considered as a strong entity and the child is a weak entity.

In ER modeling, notation for weak entity is given below.

Attributes

Properties/characteristics which describe entities are called attributes.

In ER modeling, notation for attribute is given below.

Domain of Attributes

The set of possible values that an attribute can take is called the domain of the attribute. For example, the attribute day may take any value from the set {Monday, Tuesday ... Friday}. Hence this set can be termed as the domain of the attribute day.

Key attribute

The attribute (or combination of attributes) which is unique for every entity instance is called key attribute.

E.g the employee_id of an employee, pan_card_number of a person etc.If the key attribute consists of two or more attributes in combination, it is called a composite key.

In ER modeling, notation for key attribute is given below.

Simple attribute

If an attribute cannot be divided into simpler components, it is a simple attribute.

Example for simple attribute : employee_id of an employee.

Composite attribute

If an attribute can be split into components, it is called a composite attribute.

Example for composite attribute : Name of the employee which can be split into First_name, Middle_name, and Last_name.

Single valued Attributes

If an attribute can take only a single value for each entity instance, it is a single valued attribute.

example for single valued attribute : age of a student. It can take only one value for a particular student.

Multi-valued Attributes

If an attribute can take more than one value for each entity instance, it is a multi-valued attribute. Multi-valued

example for multi valued attribute : telephone number of an employee, a particular employee may have multiple telephone numbers.

In ER modeling, notation for multi-valued attribute is given below.

Stored Attribute

An attribute which need to be stored permanently is a stored attribute

Example for stored attribute : name of a student

Derived Attribute

An attribute which can be calculated or derived based on other attributes is a derived attribute.

Example for derived attribute : age of employee which can be calculated from date of birth and current date.

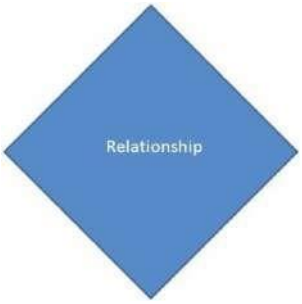
In ER modeling, notation for derived attribute is given below.

Relationships

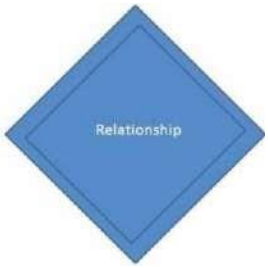
Associations between entities are called relationships

Example : An employee works for an organization. Here "works for" is a relation between the entities employee and organization.

In ER modeling, notation for relationship is given below.



However in ER Modeling, To connect a weak Entity with others, you should use a weak relationship notation as given below



Degree of a Relationship

Degree of a relationship is the number of entity types involved. The n-ary relationship is the general form for degree n. Special cases are unary, binary, and ternary ,where the degree is 1, 2, and 3, respectively.

Example for unary relationship : An employee ia a manager of another employee

Example for binary relationship : An employee works-for department.

Example for ternary relationship : customer purchase item from a shop keeper

Cardinality of a Relationship

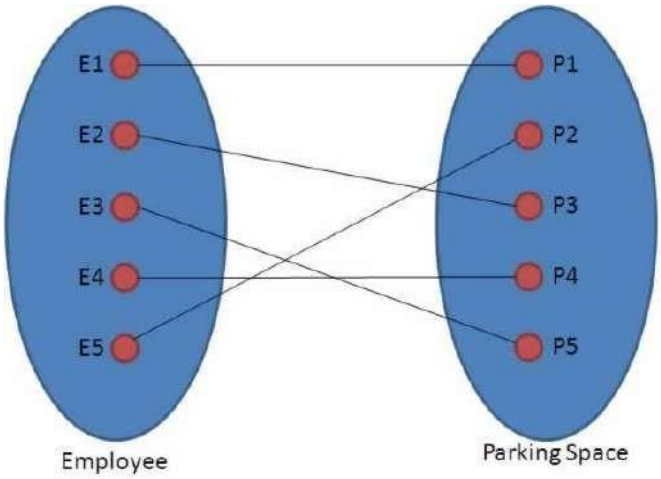
Relationship cardinalities specify how many of each entity type is allowed. Relationships can have four possible connectivities as given below.

1. One to one (1:1) relationship
2. One to many (1:N) relationship
3. Many to one (M:1) relationship
4. Many to many (M:N) relationship

The minimum and maximum values of this connectivity is called the cardinality of the relationship

Example for Cardinality – One-to-One (1:1)

Employee is assigned with a parking space.

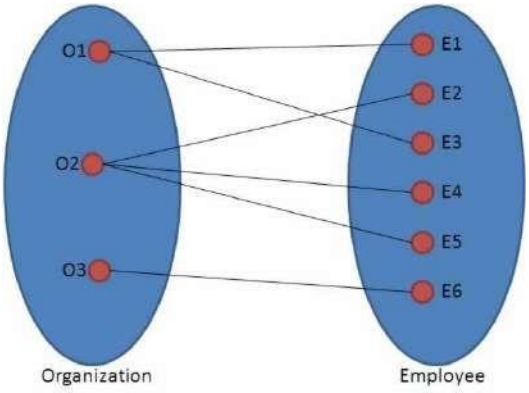


One employee is assigned with only one parking space and one parking space is assigned to only one employee. Hence it is a 1:1 relationship and cardinality is One-To-One (1:1)

In ER modeling, this can be mentioned using notations as given below

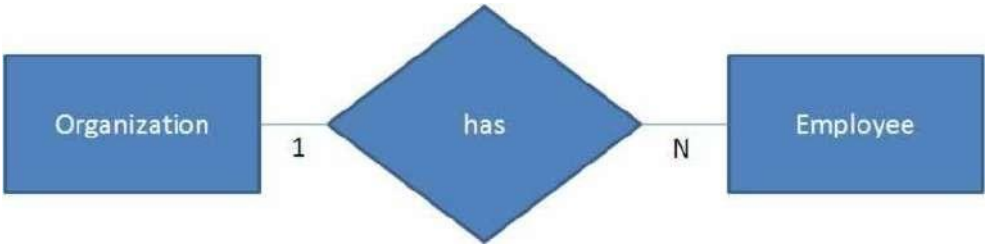
Example for Cardinality – One-to-Many (1:N)

Organization has employees



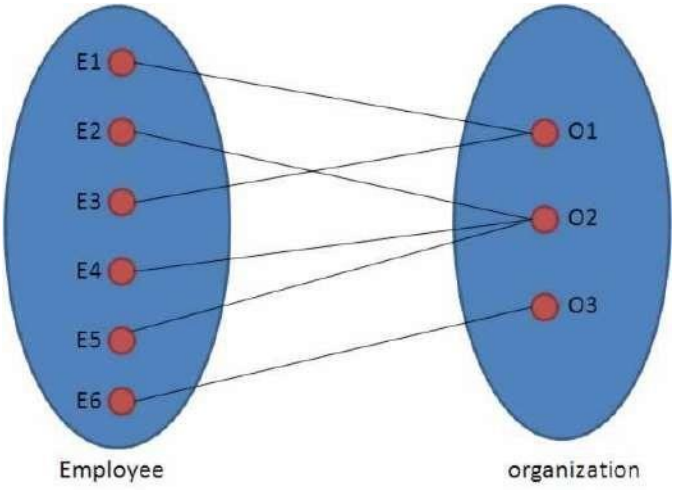
One organization can have many employees , but one employee works in only one organization. Hence it is a 1:N relationship and cardinality is One-To-Many (1:N)

In ER modeling, this can be mentioned using notations as given below



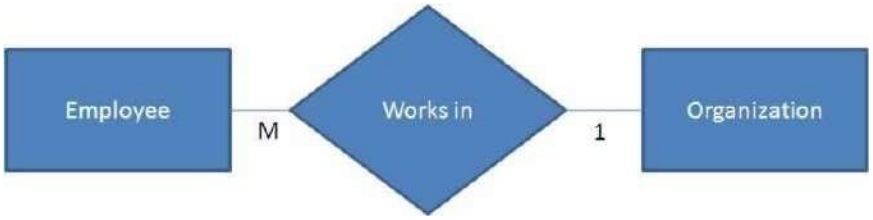
Example for Cardinality – Many-to-One (M :1)

It is the reverse of the One to Many relationship. employee works in organization



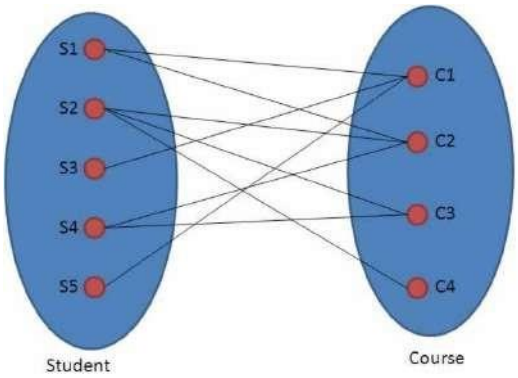
One employee works in only one organization But one organization can have many employees. Hence it is a M:1 relationship and cardinality is Many-to-One (M :1)

In ER modeling, this can be mentioned using notations as given below.



Cardinality – Many-to-Many (M:N)

Students enrolls for courses



One student can enroll for many courses and one course can be enrolled by many students. Hence it is a M:N relationship and cardinality is Many-to-Many (M:N)

In ER modeling, this can be mentioned using notations as given below

Relationship Participation

1. Total

In total participation, every entity instance will be connected through the relationship to another instance of the other participating entity types

2. Partial

Example for relationship participation

Consider the relationship - Employee is head of the department.

Here all employees will not be the head of the department. Only one employee will be the head of the department. In other words, only few instances of employee entity participate in the above relationship. So employee entity's participation is partial in the said relationship.

However each department will be headed by some employee. So department entity's participation is total in the said relationship.

Advantages and Disadvantages of ER Modeling (Merits and Demerits of ER Modeling)Advantages

1. ER Modeling is simple and easily understandable. It is represented in business users language and it can be understood by non-technical specialist.
2. Intuitive and helps in Physical Database creation.
3. Can be generalized and specialized based on needs.
4. Can help in database design.
5. Gives a higher level description of the system.

Disadvantages

1. Physical design derived from E-R Model may have some amount of ambiguities or inconsistency.
2. Sometime diagrams may lead to misinterpretations

UNIT2

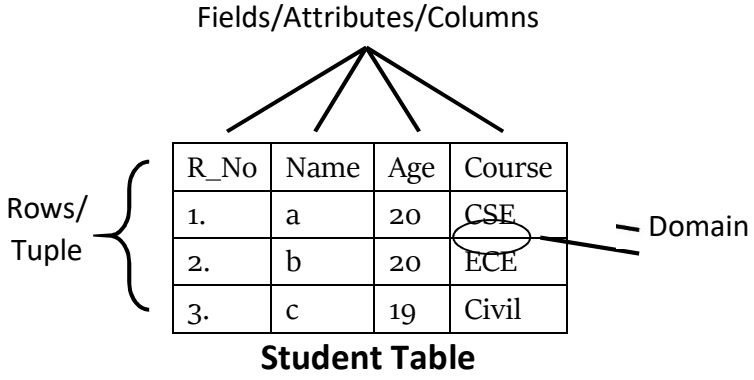
UNIT - II Introduction to the Relational Model: Integrity constraint over relations, enforcing integrity constraints, querying relational data, logical data base design, introduction to views, destroying/altering tables and views. Relational Algebra, Tuple relational Calculus, Domain relational calculus.

- Relational Model

The relational model is the popular data model used in logical design of the database. The main construct for relational model is "**Relations"**. Each relation is represented in 2 ways:

1. **Relational Schema** 2. **Relational Instance**

Consider a relation student



R_No	Name	Age	Course
1.	a	20	CSE
2.	b	20	ECE
3.	c	19	Civil

Student Table

1. The relational schema for the given relation is student (R_No, Name, Age, Course)
2. The relational instance for the given relation is 3 rows/tuples.

- Basic terminology

1. **Relation:**
   A relation is also known as a table.
2. **Attribute**
   The column of a reaction is known as attributes or fields.
3. **Domain**
   The type of values allowed for an attribute is known as domain.
4. **Degree of relation**
   The no. of attributes of a relation is known as degree of relation.
5. **Cardinality of a relation**
   The no. of records of a relation is known as cardinality of a relation
   In given table, cardinality of students = 3

- Integrity constraints over relations

Integrity constraints (IC) is a condition specify on the database schema that restrict certain data to be stored in the database instance. The integrity constraints are specified and enforced in that it allows only legal instance to be stored in the database.

- Legal instance

A legal instance is an instance that satisfies all the IC's specified on the database schema.

- Key constrains

A key constraint is a statement that a minimal set of attributes uniquely identify a record in a relation.

- Types of key constraints:

1. Candidate key 2. Primary key 3. Super key 4. Alternate key or secondary key
5. Foreign key 6. Composite key

1. Candidate key:

Candidate key is a minimal set of attributes that uniquely identifies a record in a relation. Consider a following relation

S_No	S_Name	Phone	Age
1	a	9999	20
2	b	9881	19

Student

S_No	C_No	Course
1	10	IT
2	20	CSE

Student_Course

From student relation, the candidate key may be

a) (S_No, Phone) - CK
b) (S_No, S_Name, Phone) - CK

A relation may contain any no. of candidate key. A candidate key is simply called as 'Key'.

2. Primary key

A primary key is a column or combination of columns that uniquely identifies a record in a relation

Condition for primary key

i) It will not allow duplicate values.
ii) It will not accept null values.

A relation may contain any no. of candidate keys out of which one is primary key. Therefore, a relation contains a single primary key.

Ex: Student - S_No - PK
Student_Course - S_No, C_No)

3. Composite key

When a primary key is defines using a combination of columns. It is known as composite primary key

Ex: Student □ S_No, Phone
Student_Course □ S_No, C_No

4. Super key

The set of attributes which uniquely identifies a record in a relation is known as a super key. Adding 0 or more attributes to candidate key will generate a super key.

Ex: Student □ (S_No, S_Name)
□ (S_No, S_Name, Age)

5. Alternate Key (Secondary key)

The candidate key other than primary key is known as alternate key.

Ex:

i) Student □ (S_No, S_Name) - candidate key
Alternate: S_Name
ii) (S_No, S_Name, Phone) - CK
Alternate □ (S_Name, Phone)

6. Foreign key (Referential key)

Sometimes, the information in one table is related to the information in another table. To establish the relation among the tables, we use a constraint known as foreign key. It is also known as "The referential key". It establishes the parent - child relationship among the tables.

Ex: In the above relations, Student is an original & Student_Course is a referential relation.

- General constraints

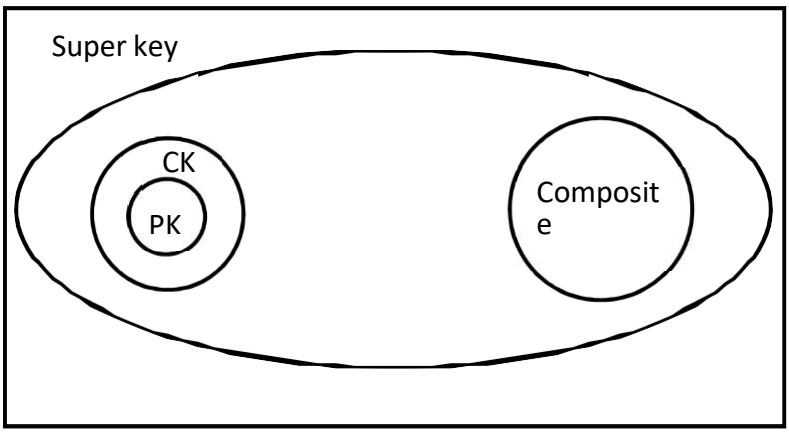
1. Table constraints 2. Assertions

1. Table constraints

These constraints are related to a single table. This constraint is defined in the table definition.

2. Assertion

These are the constraints related to multiple tables, the definition of assertions constraints is separated from table constraints



Relation among keys

- Enforcing integrity constraints

Intensity constraints are the rules or conditions specified on the tables and it will restrict incorrect data to be inserted into the table. The integrity constraints are specified and enforced at different times

1. During defining the database schema (Table definition), the integrity constraints are specified.
2. While the database application is running different integrity constraints will be enforced which causes due to the violations.

The operations such as insertion, deletion and updation must be rejected if they found to be violating the constraints specified under table.

Consider the following relation employee, department

E_No	E_Name	E_Sal	Age	D_No
1	Ravi	1000	25	10
2	Mohan	2000	26	20

D_No	D_Name	Location
10	IT	Hyd1
20	CSE	Hyd2

Ex1: Consider insertion of new record into employee table

a. insert into emp values (3, 'aa', 3000, 25, 20);

The insertion of this record is accepted as it satisfies the constraints specified on the table.

E_No	E_Name	E_Sal	Age	D_No
1	Ravi	1000	25	10
2	Mohan	2000	26	20
3	aa	3000	25	20

b. insert into emp values (2, 'bb', 4000, 21, 10);
This insertion will be rejected because the primary key is violated.

Ex2:

a. insert into dept values (30, 'Civil', 'Hyd3');
The insertion of this record into dept table is accepted as it satisfies the constraints specified on the table.

D_No	D_Name	Location
10	IT	Hyd1
20	CSE	Hyd2

b. insert into dept values (40, 2000, 'Hyd3');
The insertion of second record into dept is rejected because it generates the violation of domain constraint.

Ex3: insert into emp values (4, 'bb', 4000, 21, 50);
The insertion of this record into emp table is rejected because it causes violation of foreign key constraint.

Ex4: delete from dept where D_No=20;
The delete operation under dept table is rejected because it violates foreign key constraints.

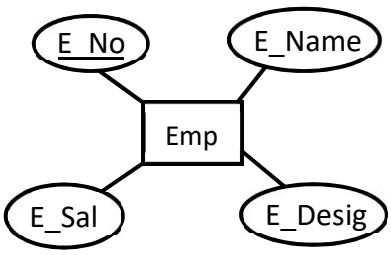
- Logical Database Design

Logical database design is the process of mapping or translating the conceptual design (ER diagrams) into relational model (relation or table). The logical database design has several concepts which include

1. **Mapping of entity set into tables**
2. **Mapping relational sets (without constraints) into table**
3. **Mapping relational sets with participation constraints into table**
4. **Mapping relational sets with key constraints into table**
5. **Mapping weak entity into table**
6. **Mapping weak class into table**
7. **Mapping ER diagram with aggregation into table**

1. Mapping of entity set into tables

Consider an entity set emp



Procedure to map entity set into table

a. Create table for an entity
b. The attribute of an entity will become attributes of a table

c. Key attribute of entity will become primary key for the table

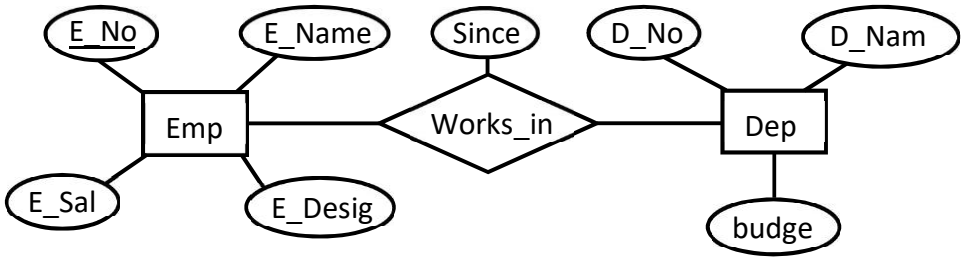
EMP Relation

E_No	E_Name	E_Sal	E_Desig

create table Emp (E_No int, E_Name char (25), E_Sal int, E_Desig char (25), primary key (E_No));

2. Mapping the relationship set (without constraints) into table

Consider the ER diagram given

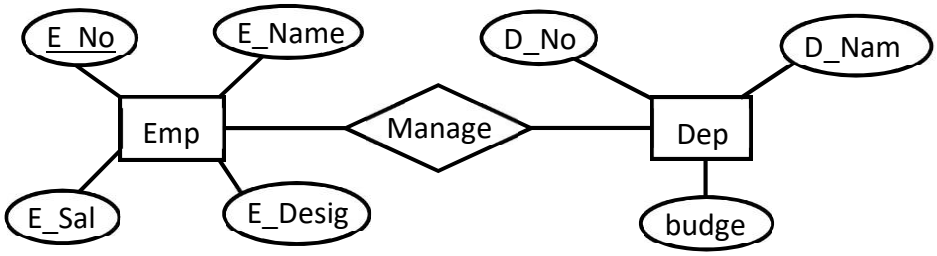


Mapping procedure

a. Create a table for relationship set (Works_in).
b. Add all primary key of entity set as attributes of the table (E_No, D_No).
c. Add the own attributes of relationship as the attribute of the table (Since).
d. Declare a primary key using all the key fields of entity set.
e. Declare a foreign key for all the field of entity set.

3. Mapping relationship set with key constraint into table

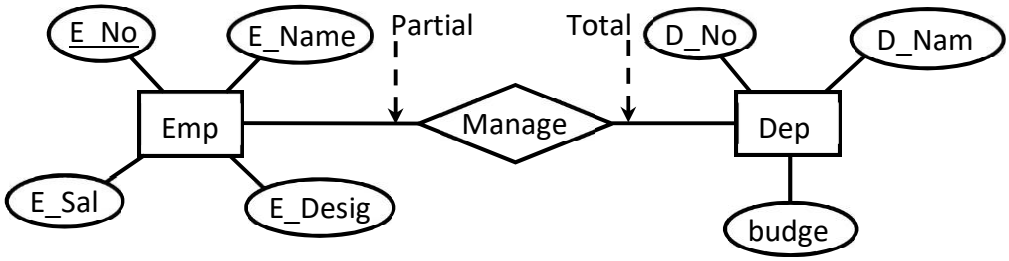
Consider the ER diagram



Mapping procedure

a. Create a table for the relationship set (manages).
b. Add all the key attributes of entity set to attributes of table
c. Add own attributes of relationship set to the table
d. Declare a primary key using the key field from source entity (D_No).
e. Declare a foreign key for key fields of source & target entity (D_No, E_No).

4. Mapping relationship set with participation of constraints

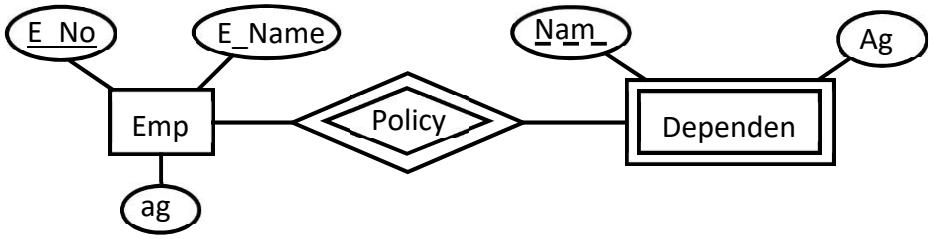


Mapping procedure

a. Create tables from source and target entity as usual

b. Add every key field of target entity in the source entity.
c. Declare these field as not null.
d. Declare these keys as foreign key.
create table Dept_Manager (D_No int, E_No int not null, primary key (D_No), foreign key (E_No) references Emp (E_No)).

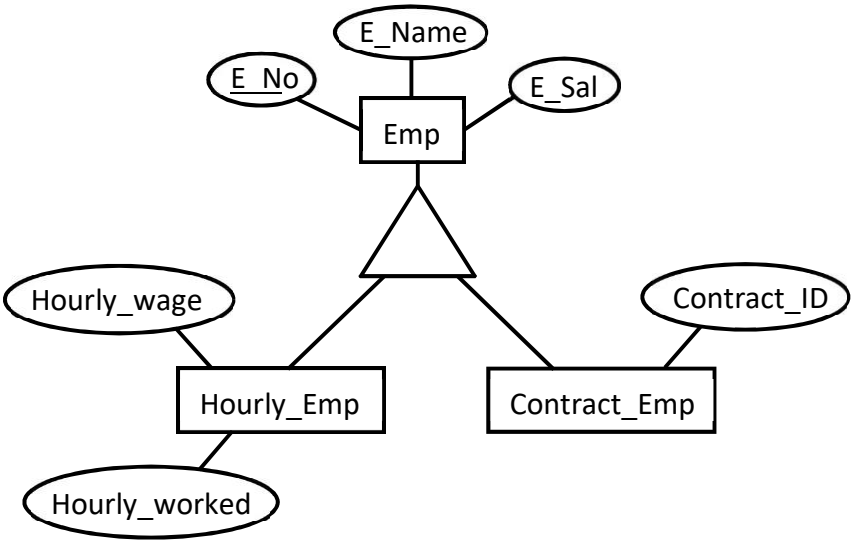
5. Translation weak entity into tables



Mapping procedure

a. Create a table for policy relationship (Dependent_policy) total participation.
b. Include the key attribute of employee and partial key of the dependent entity set along with its own attributes.
c. Declare a primary key using key attribute and partial key combination.
d. Declare a foreign key for target entity set.

6. Translating class hierarchies into table



Translating class hierarchy to tables follows 2 approaches

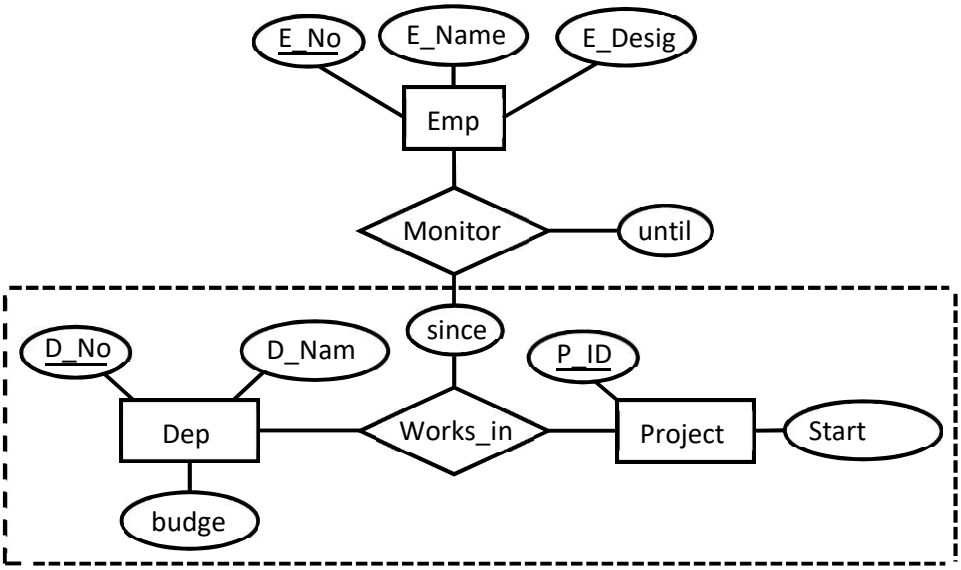
Method I

a. Emp (E_No, E_Name, E_Sal)
b. Hourly_emp (Hourly_wages, Hourly_worked)
c. Contract_emp (Contract_ID)

Method II

a. Hourly_emp (E_No, E_Name, E_Sal, Hourly_wages, Hourly_worked)
b. Contract_emp (E_No, E_Name, E_Sal, Contract_ID)

7. Translating ER Diagram with aggregation into table



create table monitors (E_No int, D_No int, P_ID int, primary key (E_No, D_No, P_ID), foreign key (E_No) references Emp (E_No), foreign key (D_No) references Dept (D_No), foreign key (P_ID) references Project (P_ID));

- Introduction to views

Sometimes, the users of the database are interested to work with only part of the data from the database. This is provided by a concept known as views.

A view is an alternate way of representing data present in one or more tables. A view is a virtual table or logical table or derived table whose data is derived from the original table. A view can include some columns or all columns from one or more tables.

- Working with views

- Syntax for creating views

> create view viewname (field1, field2, ..., fieldn)
> as
> select
> from table1, table2, ..., tablen
> where condition;

Ex: Create an Emp_DeptV using the relation employee, department

create table Emp (E_No int, E_Name char (25), E_Sal int, D_No int, primary key (E_No), foreign key (D_No) references Dept (D_No));

create table Dept (D_No int, D_Name char (25), Location char (25), primary key (D_No));

insert into Dept values (10, 'IT', 'Hyd1'), (20, 'CSE', 'Hyd2'), (10, 'EE', 'Hyd3');

insert into Emp values (1, 'aa', 1000, 10), (2, 'bb', 2000, 20), (3, 'cc', 3000, 10);

E_No	E_Name	E_Sal	D_No
1	aa	1000	10
2	bb	2000	20
3	cc	3000	10

D_No	D_Name	Location
10	IT	Hyd1
20	CSE	Hyd2
30	EE	Hyd3

i. Creating view: Emp_DeptV

> create view Emp_DeptV
> as

```
> select e1. E_No, e1. E_Name, d1. D_No, d1. D_Name
> from Emp e1, Dept d1
> where e1. D_No=d1.D_No;
> desc Emp_DeptV;
```

Field	Type	Null	Key	Default	Extra
E_No	int (11)	NO		NULL	
E_Name	char (25)	YES		NULL	
D_No	int (11)	NO		NULL	
D_Name	char (25)	YES		NULL	

```
> select * from Emp_DeptV
```

E_No	E_Name	D_No	D_Name
1	aa	10	IT
2	bb	20	CSE
3	cc	30	EE

View are dynamic in nature that mean the modifications performed on views is reflected back on original table and vice versa

ii. Updating view

```
> update Emp_DeptV set E_Name = 'Ankit' where D_No = 20;
> select * from Emp_DeptV;
```

E_No	E_Name	D_No	D_Name
1	aa	10	IT
2	Ankit	20	CSE
3	cc	30	EE

```
> select * from Emp;
```

E_No	E_Name	E_Sal	D_No
1	aa	1000	10
2	Ankit	2000	20
3	cc	3000	10

```
> update Dept set D_Name = 'Mech' where D_No = 30;
> select * from Dept;
```

D_No	D_Name	Location
10	IT	Hyd1
20	CSE	Hyd2
30	Mech	Hyd3

```
> select * from Emp_DeptV;
```

E_No	E_Name	D_No	D_Name
1	aa	10	IT
2	Ankit	20	CSE
3	cc	30	Mech

iii. Altering view (adding a column) - E_Sal

```
> alter view Emp_DeptV
> as
> select e1. E_No, e1. E_Name, e1. E_Sal, d1. D_No, d1. D_Name
> from Emp e1, Dept d1
> where e1. D_No = d1. D_No;
> desc Emp_DeptV;
```

Field	Type	Null	Key	Default	Extra
E_No	int (11)	NO		NULL	
E_Name	char (25)	YES		NULL	
E_Sal	int (11)	YES		NULL	
D_No	int (11)	NO		NULL	
D_Name	char (25)	YES		NULL	

```
> select * from Emp_DeptV;
```

E_No	E_Name	E_Sal	D_No	D_Name
1	aa	1000	10	IT
2	Ankit	2000	20	CSE
3	cc	3000	30	Mech

iv. Drop view

Syntax: drop view viewname;

- Advantages of Views:

1. Views provide security that means the users are working with the past of the database rather than using the entire database. Therefore, the original table will not be disturbed and hence secure from unauthorized users.
2. View provide logical data independence in which data and relationship among the data is maintained even though modification at the external schema has been done

- Updateable views:

Whenever a view is updated, the result should be reflected in the original table and vice versa but all updation are not allowed on views.

An updatable view is a view which is derived from a single table and it should hold the following conditions:

1. Aggregate functions should not be used in query.
2. Distinct keyword should not be allowed.
3. Group by and having clause will not be allowed.

- Relational Model

Relational model is a popular data model used for logical database design. There are 2 formal query language associated with the relational model.

1. Relational Algebra 2. Relational Calculus

1. Relational Algebra

Relational algebra is a procedural query language which uses collection of operators to write different queries.

2. Relational Calculus

Relational Calculus is a non – procedural query language which is based on predicate calculus.

- Relational Algebra

1. It is a procedural query language (the user has to specify what data he require along with the procedure).
2. The relational algebra queries are a combination of operators.
3. Each operator takes one or more relations as arguments and returns a relation as result or output.
4. It uses the operation like relational operators (<, >, <=, >=, =, !=) and logical connectives (and, or, not) to write various composite and complex queries.

- Fundamental operations of relational algebra

1. Selection (σ) 2. Projection (π) 3. Set operations (Union, Intersection, Difference)
4. Rename 5. Division 6. Joins

1. Selection (σ)

a. It is a unary operator.

b. It is represented with the symbol σ.

c. It is used to select the subset of tuples from a relation that matches a given condition.

Syntax: $\sigma_{<Condition_Statement>}$relation_name

Condition statement has 2 formats

a. Operand operator constant
b. Operand operator operand

Consider the 2 relation Emp, Manager to demonstrate relation algebra operator.

E_No	E_Name	E_Sal
1	aa	1000
2	bb	2500
3	cc	4500
4	dd	6000

ID	Name	D_No
1	Raj	10
2	bb	20
3	aa	30

//display Emp where Sal is greater than 2000Ex: $\sigma_{E_Sal>2000}$Emp

2	bb	25000
3	cc	45000
4	dd	6000

2. Projection (π)

a. It is a unary operator.

b. It is represented by symbol π.

c. It is used to select the subset of attributes from the given relation.

Syntax: $\pi_{\text{colname 1, 2, ...}}$Table_Name

//display the names and salaries of employee from Emp relation

Ex1: $\pi_{\text{E\_Name, E\_Sal}}$ Emp

E_Name	E_Sal
aa	1000
bb	25000
cc	45000
dd	6000

//Display names of all employees where salary is greater than 2000

Ex1: $\pi_{\text{E\_Name}}(\sigma_{\text{E\_Sal>2000}})$ Emp

E_Name	E_Sal
aa	1000
bb	25000
cc	45000
dd	6000

3. Set operations (Union, Intersection, Difference)

a. Union (∪)

$\pi_{\text{E\_Name}}$ Emp ∪ π_{Name} Manager

E_Name
aa
bb
cc
dd
Raj

Union

E_Name
aa
bb
cc
dd
Raj
bb
aa
cc

Union All

b. Intersection (∩)

$\pi_{\text{E\_Name}}$ Emp ∩ π_{Name} Manager

E_Name
aa
b
b

c. Difference (-)

π_{E_Name} Emp – π_{Name} Manager

E_Name
dd

Raj

4. Rename

Rename operator is used to give the alias name or temporary name to a relation as well as to the attributes of a relation. It is denoted with the symbol ρ-rho which is a Greek letter.

Syntax: ρ_x ® R – Relation

x – Alias/temporary name

$\rho_x (A_1A_2A_3 \ldots A_n)$ ® R – Relation

$A_1A_2A_3 \ldots A_n$: New name of attribute

x – Alias name for relation

5. Division

The division operator is used in special kind of queries that include the phase for all. It is denoted with the symbol **(/)**.

Consider 2 relations R, S. R contains attributes (a, b) and S contains attribute (b).

$\frac{R}{S}$ is given as $\frac{(a,b)}{b} = a$

For all values of A in relation R and for each value of B in relation S there is a tuple (A, B) in R.

Ex: Consider Relation table

A =

S_No	P_No
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1 =

P_No
p2

B2 =

P_No
p2
p4

B3 =

P_No
p1
p2
p4

A/B1 =

S_No
s1
s2
s3
s4

A/B2 =

S_No
s1
s4

A/B3 =

S_No
s1

6. Joins

Join operation is used to combine 2 relations like cross product but finally remove the duplicates. There are 3 types of joins

a. Conditional Joins ($\bowtie_C$) b. Equi Joins ($\bowtie_=$) c. Natural Joins ($\bowtie$)

Consider the relations Dept, Project to demonstrate joins

D_Name	D_No
IT	10
CSE	20
Mech	30

Dept

D_No	P_No	P_Name
10	1	Sales
20	2	HR

Project

a. Conditional Joins ($\bowtie_C$)

It returns a relation that contain a set of rows from cross product (X) such that each row satisfies a given condition. It is denoted by $\bowtie_C$.

Consider 2 relations R1, R2. The conditional join of R1, R2 is given below

$$R1 \bowtie_C R2 = \sigma_C (R1 \text{ X } R2)$$

Ex: Find the conditional join of relation dept, projects where Dept. D_No < Project. D_No

Dept $\bowtie_C$ Project = σ_C (Dept x Project)

Step 1: Dept x Projects

D_Name	D_No	D_No	P_No	P_Name
IT	10	10	1	Sales
IT	10	10	2	HR
CSE	20	20	1	Sales
CSE	20	20	2	HR
Mech	30	30	1	Sales
Mech	30	30	2	HR

Step 2: σ_C (Dept x Project)

$\sigma_{\text{Dept. D\_No<Project. D\_No}}$ (Dept x Project)

D_Name	D_No	D_No	P_No	P_Name
IT	10	20	2	HR

b. Equi Joins ($\bowtie_=$)

It is similar to conditional join except the condition used to select the record. Here we use equality operator to join 2 relations. The result of equi join contain the attributes of relation A followed by relation B excluding duplicate attributes.

$$R1 \bowtie_= R2 = \sigma_= (R1 \text{ X } R2)$$

Ex: Consider same Dept and Project relation

Then, Dept $\bowtie_=$ Project = $\sigma_=$ (Dept x Project)

Step 1: Dept x Project

D_Name	D_No	D_No	P_No	P_Name
IT	10	10	1	Sales
IT	10	10	2	HR
CSE	20	20	1	Sales
CSE	20	20	2	HR
Mech	30	30	1	Sales
Mech	30	30	2	HR

Step 2: $\sigma_=$ (Dept x Project)

$\sigma_{\text{Dept. D\_No=Project. D\_No}}$ (Dept x Project)

D_Name	D_No	D_No	P_No	P_Name
IT	10	10	1	Sales
CSE	20	20	2	HR

c. Natural Joins ($\bowtie$)

It is a special case of equi join in which the equality conditions are specified on all columns. It doesn't have 2 columns with the same name.

Ex: Consider the relations Boats, Sailors & Reserve table to demonstrate relational algebra.

S_ID	S_Name	S_Age	Rating
10	aa	20	1
20	bb	21	2
30	cc	23	4

Sailors

B_ID	B_Name	Colour
101	Interlake	Blue
102	Duster	Green
103	Interlake	Red

Boats

S_ID	B_ID	Day
10	101	10/09/18
20	102	10/09/18
30	102	09/09/18

Reserves

Q1: Find names of Sailors who have reserved boat 102

$\pi_{S\_Name}((\sigma_{B\_ID = 102} Reserves) \bowtie Sailors)$

S_Name		S_ID	B_ID	Days
bb		20	102	10/09/18
cc		30	102	09/09/18

Q2: Find name of Sailors who have reserved a red boat

$\pi_{S\_Name}((\sigma_{Colour = 'Red'} Boats) \bowtie Reserves \bowtie Sailors)$

Empty Set

B_ID	B_Name	Colour
103	Interlake	Red

Empty set

Q3: Find the colour of the Boat reserved by bb

$\pi_{Colour}((\sigma_{S\_Name = 'bb'} Sailors) \bowtie Reserves \bowtie Boats)$

Colour
Green

S_ID	S_Name	S_Age	Rating
20	bb	21	2

S_ID	B_ID	Day
20	102	10/09/18

Q4: Find names of Sailors who have reserved a red or a green boat

$\pi_{S\_Name}((\sigma_{Colour = 'Red'} Boats \cup \sigma_{Colour = 'Green'} Boats) \bowtie Reserves \bowtie Sailors)$

S_Name
bb

B_ID	B_Name	Colour
102	Duster	Green
103	Interlake	Red

S_ID	B_ID	Day
20	102	10/09/18
30	102	09/09/18

Q5: Find names of Sailors who have reserved at least one boat

$\pi_{S\_Name}(Sailors \bowtie Reserves)$

- Relational Calculus

Relational calculus is a non-procedural query language or declarative language where the user has to specify what he require without worrying about the procedure. Relational calculus is of 2 types.

1. **Tuple Relational Calculus (TRC)** 2. **Domain Relational Calculus (DRC)**

1. Tuple Relational Calculus

Tuple Relational calculus uses tuples as the values to the variables. Each tuple in TRC is expressed by a TRC expression. A TRC expression has the following form.

Syntax:

a. $\{t_1A_1, t_2A_2, ..., t_nA_n | Q\}$

$t_1, t_2, ..., t_n$ – tuple variable

$A_1, A_2, ..., A_n$ – Attribute of tuple variable

Q – Condition/Formula

b. $t | f(t)$

t – Tuple variable

f(t) – Formula involving Tuple variable t

A TRC uses tuples from relational database by writing/using predicate calculus expression.

Note: TRC is shortly influenced by SQL.

Consider a student relation given below

S_No	S_Name	D_No	Gender
1	aa	10	Male
2	bb	20	Female
3	cc	20	Female

Student

Q1: Find S_No, S_Name from D_No = 20

{t. S_No, t. S_Name| Student(t) ⍰ t. D_No = 20}

S_No	S_Name
2	bb
3	cc

Q2: Find the names of male students in D_No = 20

{t. S_Name| Student(t) ⍰ t. D_No = 20 ⍰ t. Gender = 'Male'}

S_Name
cc

Consider the following relation: Depositor, Borrower, Loans, Customer, Account, Branch to demolish TRC queries

C_Name	Account_No
Ravi	1001
Raju	1002

Depositor

C_Name	Loan_No
Ravi	100
Raju	101
Anil	102

Borrower

Loan_No	B_Name	Amount
100	Kompali	1000
101	Kachiguda	2000
102	Nagole	3000
103	Dilshuknagar	4500

Loan

C_Name	Address
Ravi	Kompali
Ramu	Kachiguda
Anil	Nagole
Ankit	Dilshuknagar

Customer

Account_No	B_Name	Balance
1001	Kompali	1000
1002	Kachiguda	2500
1003	Nagole	4000

Account

B_Name	City
Kompali	Hyderabad
Kachiguda	Hyderabad
Nagole	Hyderabad

Branch

Q1: Find loan details of loan amount > 2000

{t.| Loan(t) ∧ t. Amount>2000}

Loan_No	B_Name	Amount
102	Nagole	3000
103	Dilshuknagar	4500

Q2: Find the names of all customers who have a loan from the branch 'Kachiguda'

{b. C_Name| Borrower(t) ∧ L(Loan(L) ∧ L. Loan_No = b. Loan_No ∧ L. B_Name = 'Kachiguda'}

Q3: Find the customers who have account or loan or both

{t| Customer(t) ∧ ∃ d(Depositor(d) ∧ d. C_Name = t. C_Name) ∨ ∃ b(Borrower(d) ∧ b. C_Name = t. C_Name)

2. Domain Relational Calculus

A DRC will operate at the domain of a variable. It uses the domain values for the variables that DRC is strongly influenced by QBE (Query By Example).

A DRC query will have the following form

Syntax: $\{x_1, x_2, x_3, \ldots, x_n | \underbrace{Q(x_1, x_2, x_3, \ldots, x_n)}\}$

Domain Variable Formula or condition

Consider the relation Emp, Dept as given

A B C D E F

F_Name	L_Name	E_ID	Salary	D_No	Address
Raj	Kumar	1	1000	10	Kotapet
Anil	M	2	2000	20	Abids
Ravi	Verma	3	4000	10	Dilshuknagar

X Y Z

D_No	D_Name	M_ID
10	IT	1
20	HR	4
40	CSE	2

Q1: Find the name and address of employees whose E_Name = 'Raj Kumar'

{ABF| ∃ C ∃ D ∃ E Emp (ABCDEF) ∧ (A = 'Raj') ∧ (B = 'Kumar')}

Or

{ABF| Emp (ABCDEF) ∧ (A = 'Raj') ∧ (B = 'Kumar')}

Or

{ABF| Emp ('Raj', 'Kumar', C, D, E, F)}

Q2: Find names of employees working in D_No = 10

{AB| Emp (ABCDEF) ∧ E = 10}

Q3: Find names of employees who are not managers

{AB| ∃ C Emp (ABCDEF) ∧ ~ (∃ Z (Dept (XYZ) ∧ C = Z))}

Q4: Find names of employees working in HR department

{AB| ∃ E Emp (ABCDEF) ∧ ∃ X (Dept (XYZ) ∧ E = Z ∧ Y = 'HR')}

UNIT 3

- **UNIT - III SQL: QUERIES, CONSTRAINTS, TRIGGERS: form of basic SQL query, UNION, INTERSECT, and EXCEPT, Nested Queries, aggregation operators, NULL values, complex integrity constraints in SQL, triggers and active data bases. Schema Refinement: Problems caused by redundancy, decompositions, problems related to decomposition, reasoning about functional dependencies, FIRST, SECOND, THIRD normal forms, BCNF, lossless join decomposition, multi-valued dependencies, FOURTH normal form, FIFTH normal form**
-
-
- **Forms of basic SQL Query**
  - **Query:**

    A query is a question which will retrieve data from the tables or database. The result of a query is a relation or tables.

    The relational database consists of many relations where each relation has a unique name. To interact with the database, we are a query language known as SQL (Structured Query Language). The SQL enables to write different queries and retrieve data from the tables.

    The basic SQL query consists of the following 3 clauses:

1. Select Clause 2. From Clause 3. Where Clause

1. Select clause

The select clause will provide the attributes to be displayed in the resultant table.

2. From clause

The from clause indicates the table or tables from which attributes are selected

3. Where clause

The where clause indicates the condition based on which data is selected from the table.

- Examples of basic SQL queries

1. Consider the following schema Employee (E_ID, E_Name, E_Sal, E_Age, Ph_No, D_No)
   Department (D_No, D_Name, Location, M_ID)

Query 1: Find the names of all the employees who are working in the HR department.

Query 2: Find E_ID, E_Name, D_Name from Employee, Department where D_No = 20.

Query 3: List all the information about employees whose E_Sal >= 1500.

Employee Table ↗F.K

E_ID	E_Name	E_Sal	E_Age	Ph_No	D_No
1	a	1000	20	9599	10
2	b	1600	21	8658	20
3	c	3000	19	9988	30

Department Table

D_No	D_Name	Location	M_ID
10	IT	Hyd1	1001
20	HR	Hyd2	1002
30	CSE	Hyd3	1003
40	EE	Hyd4	1004

Query 1: select e1. E_Name from Employee e1 Department d1 where e1. D_No = e2. D_No and d1. D_Name = 'HR';

E_Name
b

Query 2: select E_ID, E_Name, D_Name, from Employee e1 Department d1 where e1. D_No = d1. D_No and d1. D_No = 20;

E_ID	E_Name	D_Name
2	b	HR

Query 3: select * from Employee where E_Sal >= 1500;

E_ID	E_Name	E_Sal	Age	Ph_No	D_No
2	B	2000	21	8658	20
3	c	3000	19	9988	30

- Aggregate operators (Function)

Aggregate operators are the operators which work on set of values and return a single value as the output. The most commonly used aggregate operators include:

1. count () 2. max () 3. min () 4. sum () 5. avg ()

Consider a sample relation emp to demonstrate the aggregate functions.

E_No	E_Name	E_Sal	Age	Desig
1	Ravi	1000	20	IT
2	Raj	2000	19	CSE
3	Arun	2500	21	IT
4	Venkat	3000	20	ME
5	Raj	3500	19	HR

1. **count ():** This function will return the no. of records from a table matching the condition.
   Syntax: select count (colname) from Table_Name;
   Ex:
   > select count (*) from Emp;

Count
5

   > select count (distinct E_Name) from Emp;

Count
4

   > select count (*) from Emp where E_Sal>2500;

Count
2

   > select count (*) as No_of_Records from Emp

No_of_Records
5

2. **max ():** This function returns max values.
   Syntax: select max (colname) from Table_Name;
   Ex:
   > select max (E_Sal) from Emp;

E_Sal
3500

   > select max (E_Sal) from Emp where age>19;

E_Sal
3000

3. **min ():** This function returns min values.
   Syntax: select min (colname) from Table_Name;
   Ex:
   > select min (Age) from Emp;

Age
19

   > select min (E_Sal) from Emp where age<20;

E_Sal
2000

4. **sum ():** This function returns sum values.

Syntax: select sum (colname) from Table_Name;

Ex:

> select sum (E_Sal) from Emp;

E_Sal
12000

> select sum (E_Sal) from Emp where age>19;

E_Sal
6500

5. **Avg ():** This function returns average of a value of a column.

Syntax: select Avg (colname) from Table_Name;

Ex:

> select Avg (E_Sal) from Emp;

E_Sal
2400

> select Avg (E_Sal) from Emp where age>19;

E_Sal
216

- Null Values:

A null value is a value which is unknown or unavailable. Sometimes, the users are in such cases, DBMS will treat those unknown value or unavailable. A null value is not equal to zero or space.

Consider a sample relation employee to demonstrate null value.

E_No	E_Name	E_Sal	D_No
1	aa	1000	10
2	bb	2000	20

1. insert into Emp values (3, 'cc', 30);
   This insertion will be rejected because no. of columns not match
2. insert into Emp values (3, 'cc', NULL, 30);
   insert into Emp values (4, NULL, 3500, 20);

E_No	E_Name	E_Sal	D_No
1	aa	1000	10
2	bb	2000	20
3	cc	NULL	30
4	NULL	3500	20

- Comparing null values

1. Generally, for comparison statements, we use two valued logic (true or false) as the result.
2. When comparing NULL values with the actual values, the two valued logic will not be used. Hence, we require a three-value logic (true false and unknown) for the result.

- Logical connectives AND, OR, NOT with null values

The logical connectives AND, OR, NOT with null values must be used with a 3-valued logic i.e. True, false unknown.

- Disallowing/Restricting null values

There are 2 ways to disallow or restrict null values to be inserted in the table

1. Declare a column or columns as primary key.
2. Declare a column or columns as not NULL.

Ex: create table Emp (E_No int primary key, E_Name char (25) not null, E_Sal int not null);

Nested Queries & Co - related Nested Queries

1. Nested Query

A query inside another query is known as a sub-query. A collection of sub-queries will form Nested sub-query. In nested sub-queries, the inner query will be evaluated.

Syntax: select colname from Table_Name } Outer query/Main query

where colname operates

(select colname from Table_Name where condition); // Inner query/Sub query

S_ID	S_Name	S_Age
S01	aa	20
S02	bb	19
S03	cc	19
S04	dd	22

Student

C_ID	C_Name
C01	ADS
C02	DBMS
C03	JAVA

Course

S_ID	C_ID
S01	C01
S02	C02
S03	C03
S04	C02

Student_Course

Q1: Find S_ID of students who enrolled course ADS or DBMS

Ans: Inner Query:

> select C_ID from Course where
> C_Name = 'ADS' or C_Name = 'DBMS';

C_ID
C01
C02

Main Query:

> select S_ID from Student_Course where C_ID in
> (select C_ID from Course where C_Name = 'ADS' or C_Name = 'DBMS');

S_ID
S01
So2
S04

Q2: Find the S_Name of students who enrolled the course DBMS or JAVA

Ans:

> select S_Name from Students where S_ID in (select S_ID from Student_Course where C_ID in (select C_ID from Course where C_Name = 'DBMS' or C_Name = 'JAVA'));

2. Co - related Nested Query

If the sub-query (inner query) is evaluated repeatedly based on the result of outer query such queries are known as co - related sub - queries. Here, outer query will be executed first based on which the inner query will be evaluated repeatedly. For co - related sub query, we use 'exist' comparison operator rather than 'in' operator.

- Triggers and active database

1. Triggers

a. A trigger is a store procedure i.e. invoked or generated automatically by the DBMS whenever the database is updated or modified. Triggers are generally automatically as a response to the change made on the database.

b. Triggers are managed by database administrator. The general format of trigger consists of the following

i. An event ii. A condition iii.An action

i. The **event** describes the operation performed on the database which leads to activation of trigger Ex: insert, update, delete

ii. The **condition** specifies whether an action should be performed or not. If the condition is true, the action part will be executed and if the condition is false, the action will not be performed.

iii. The **action** specifies the response to be taken for the activation of a trigger. Action is the collection of statement executed as part of trigger activation.

- Types of triggers

a. Before triggers

Before triggers are invoked before an operation performed (insert, update, delete).

b. After triggers

These are invoked after an operation is performed (insert, update, delete).

c. Row-level triggers

It is invoked for each record inserted by the user

d. Statement level triggers

These triggers are invoked for each statement consist of multiple rows. Hence, it is called statement level trigger.

- Creation of triggers

Syntax:

> create trigger trigger_name
> trigger_time trigger_event
> on Table_Name
> before/after
> insert/delete/update
> for each row
> begin
{
//Set of SQL statements
}
> end

- Working with triggers

Consider the following relation: User, User_History to demonstrate triggers

ID	Name	Age	Weight	Address
1	aa	20	65	Hyd1
2	bb	19	60	Hyd2
3	cc	22	55	Hyd3
4	dd	21	62	Hyd4

Command	Type	Keyword
Insert	Before/After	New
Update	Before/After	New/Old
Delete	Before/After	Old

Ex1: //Consider a trigger: update trigger

> delimiter $$
> create Alter_Update_trigger
> after update on Users
> for each row
> begin
> insert into User_History (ID, Name, Age, Weight, Address) values (old. ID, old. Name, old. Age, old. Weight, old. Address);
> end $$
> delimiter;
> update users set Name = 'Anil' where ID = 1;
> select * from Users;
> select * from User_History;

ID	Name	Age	Weight	Address
1	Anil	20	65	Hyd1

User_History

Ex2: //Create a trigger After_Insert

```
> delimiter $$
> create After_Insert_trigger
> after insert on Users
> for each row
> begin
> insert into User_History values (new. ID, new. Name, new. Age, new. Weight, new. Address);
> end $$
> delimiter;
> insert into Users values (5, 'Ravi', 20, 65, 'Hyd5');
> select * from Users;
> select * from User_History;
```

ID	Name	Age	Weight	Address
5	Ravi	20	65	Hyd5

User_History

Ex3: //Create a trigger Before_Delete

```
> Delimiter $$
> Create Before_Delete_trigger
> Before delete on Users
> for each row
> begin
> insert into User_History values (old. ID, old. Name, old. Age, old. Weight, old. Address);
> end $$
> delimiter;
> delete from Users from ID = 2;
> select * from Users;
> select * from User_History;
```

ID	Name	Age	Weight	Address
2	bb	19	60	Hyd2

User_History

- Applications of triggers

a. Triggers will alert the users about unusual events.
b. It helps to enforce some business rules.
c. It validates the data even before updation or deletion or insertion.
d. Triggers will generate a log of events to support auditing and security checks.

- Limitation of triggers

a. Triggers increase overhead under system because it is called for every event like insert, update & delete this causes makes the system to run slow.
b. It is difficult to give triggers compared to other database objects such as indexes.

2. Active database

a. A database that contain a set of associated triggers is known as an active database.
b. A database that has an ability to immediately react to the events occurring inside as well as outside of the system is called an active database.

c. The ability to respond external events is called active behaviour.

d. The active behaviour is based on the rules known as ECA (Event Condition Action) rules.

- Designing active database

Designing an active database is very difficult task because sometimes it contains recursive triggers. The activation of such long chain of triggers and the predictable order in which DBMS will process the activated triggers which is very difficult to understand.

- Complex Integrity Constraints (IC's) in SQL

The complex integrity constraints in SQL is represented in 3 ways.

1. IC's over a single relation (table constraint)

3. IC's over multiple relations (Assertion)

1. IC's over a single relation (table constraint)

The table constraint is a constraint i.e. defined for a single relation it uses check constraint.

Ex:

> create table Sailors (S_ID int not null primary key, S_Name char (15), S_Age int, check (S_ID >= 1and S_ID <= 10));

> insert into Sailors values (11, 'aa', 20);

This insertion will be rejected because check constraint is violated.

Note: Table constraints (check constraint) cannot be implemented in MySQL.

2. Domain constraint

By using domain constraint, we can create our own domain rather than using default domain.

Ex:

> create domain ratingval integer default 1

> check (ratingval >= 1 and ratingval <= 10);

In the above statement, a domain is created with the name 'ratingval' where its source type is integer and default value are 1. The values of 'ratingval' are further restricted by using check constraint.

Ex:

> create table Sailors (S_ID int, S_Name char (15), S_Age int, Rating ratingval);

3. IC's over multiple relations (Assertion)

i. The table constraints which are associated with a single table and it will work only if the associated table is non-empty.

ii. When a constraint involves 2 or more relation, the table constraints will not work. To overcome this, situation, SQL provide a constraint known as assertion (constraints over multiple relation).

To enforce the constraint that no. of boats and no. of sailors all together should be less than 100.

> create assertion smallcuts

> check ((select count (S. Sal) from Sailors S) + (select count (B.ID) from Boats B) < 100);

Note: Assertion cannot be implemented in MySQL.

2. Domain constraint

1. Set operations (Union, Intersection, Difference)

a. Union (∪)

π_{E_Name} Emp ∪ π_{Name} Manager

U

E_Name
aa
bb
cc
dd
Raj

ni

o

n

E_Name
aa
bb
cc
dd
Raj
bb
aa
cc

Union All

b. Intersection (∩)

π_{E_Name} Emp ∩ π_{Name} Manager

E_Name
aa
b
b

c. Difference (-)

π_{E_Name} Emp – π_{Name} Manager

E_Name
dd

- Schema refinement

Schema refinement is one of the steps in DB design process. It is the process of refining the schema so that we remove redundancy from the database.

- Schema

The overall design of the database is known as the database scheme.

- Redundancy

When the same data is stored multiple times unnecessarily in the database, it leads to redundancy problem. Redundancy means duplication of the data stored at multiple location in the database.

- Problems caused by redundancy

1. Wastage of storage space 2. Inconsistency of data 3. Anomalies (Insert, update & delete)

1. Wastage of storage space

When the same information is stored multiple times in the database, it leads to wastage of storage space and accessing data from such database is time consuming

2. Anomalies

Anomalies are the problems i.e. caused due to partially planned unstructured database. There are 3 types of anomalies:

a. Insert Anomalies b. Delete Anomalies c. Update Anomalies

Consider the student info relation to demonstrate Anomalies.

S_No	S_Name	Age	D_No	Branch	HOD
1	A	20	10	CSE	Anil

2	B	19	10	CSE	xx
3	C	18	20	EE	yy
4	D	19	20	EE	yy
5	E	21	30	IT	zz

a. Insert Anomalies

Certain data cannot be inserted into database without the presence of other data.

Ex: Suppose we want to store information of civil department where no student enrolled into civil.

b. Delete Anomalies

If we want to delete some unwanted data, it causes deletion of some useful data.

Ex: If we want to delete student info of IT branch, then it causes deletion of branch info of IT.

c. Update Anomalies

If we want to update a single record of data then it must be done for all the copies of data

Ex: Suppose if we want to update HOD name for CSE department, then this updation must be reflected for all the copies of CSE HOD.

3. Inconsistency

Redundancy leads to a problem known as inconsistency which occur whenever multiple copies are not updated simultaneously.

Functional Dependencies (FD)

Let us consider a relation R with a set of attributes x, y where x, y ϵ R, then an FD is given as

E_D: x → y

determinant dependent

The FD's are used to represent the relation among attributes

- Types of FD's

1. Trivial FD 2. Non-trivial FD

1. Trivial FD

Suppose we have an FD x → y where y → x, such FD's are known as trivial FD.

Ex: AB → A

The above FD is trivial but not useful as it does not determine anything new.

2. Non-trivial FD

Suppose we have a FD x → y where y ⊈ x such FDs are known as non-trivial FDs

Ex: AB → ABC

The above FD is non-trivial as it determines a new attribute C

Note: The non-trivial FDs are mostly used to solve different normalization problems

- Reasoning about FD's

If a set of FDs are given over the relation R then several additional FD's can be derived over R only when the set of FDs given over R is satisfied.

Ex: Consider a sample relation Emp as given:

Emp (E_No, E_Name, Sal, D_No, D_Name)

FDs F: E_No → D_No

D_No → D_Name → E_No → D_Name

In the above example, we can derive a new FD i.e. E_No → D_Name from above 2 FDs.

- Closure of set of FDs

The closure of set of FDs F is given as F^+ computing closure of set of FDs to find the closure of set of FDs, we use a set of rules known 'Armstrong Axioms'.

1. Reflexivity

An FD: x → y holds where y → x.

2. Augmentation

An FD: x → y then xz → yz holds for an attribute z.

3. Transitivity

An FD x → y, y → z holds then x → z also holds.

- Additional rules to find closure of set of FDs

1. **Union**

   FD: x → y, x → z then x → yz.

2. **Decomposition**

   FD: x → yz, then x → y, x → z also holds.

Ex 1: R (A, B, C) **A → B,** **B → C**

A^+: ABC

B^+: BC

C^+: C

- Normalization

It is a systematic approach of reducing or removing redundancy from the database tables. To perform normalization, we use functional dependencies.

- Normal form

A normal form is a rule or condition that is applied sequentially on the database table to remove redundancy from the tables.

- Types of normal form

1. **First Normal Form (1NF)** 2. **Second Normal Form (2NF)** 3. **Third Normal Form (3NF)**
4. **Forth Normal Form (4NF)** 5. **Fifth Normal Form (5NF)**

6. **Boyce-Codd Normal Form (BCNF or 3.5NF)**

1. First Normal Form (1NF):

A relation R is said to be in 1NF if every attribute contains only atomic values means multiple values are not allowed for any column of a relation.

Consider a relation Student as given:

S_No	S_Name	Course
1	A	DBMS, COA
2	B	OS, CN

In the above relation, the course column contains multiple values. Hence, the relation is not in 1NF.

* Converting a relation to be in 1NF

S_No	S_Name	Course
1	A	DBMS
1	A	COA
2	B	OS
2	B	CN

The relation satisfies the condition of 1NF. Therefore, this relation is in 1NF.

2. Second Normal Form (2NF):

A relation is said to be in 2NF iff

a. It must be 1NF

b. Partial dependency should not exist.

Second Normal Form is based on the concept of partial dependency & full functional dependencies.

i. Partial dependency

When a non-prime attribute is depending part of the key, such dependencies is known as partial dependency

Consider a relation R(ABCD) holding the following FDs

AB → B, **B → C**

Candidate key $(AB)^+$ = ABCD

ii. Prime attribute

The attribute which are part of key are known as prime attributes.

Ex: A, B in above

iii. Non - prime attributes

The attributes which are not the part of the key are known as non - prime attributes.

In above relation, the FD: B → C indicates the partial dependency because the non – prime attribute C is depending on part of the key i.e. B. Therefore, the above relation is not in 2NF.

iv. Decomposing a relation to be in 2NF

I. The first decomposition must be with candidate key combination

i.e. R_1(ABD) AB → D

II. The second decomposition is

R_2(BC) B → C

The decomposed relations are not having any partial dependency. Hence, we can conclude that these relations are in 2NF.

3. Third Normal Form (3NF):

A relation R is said to be in 3NF iff

a. It must be 1NF & 2NF

b. No transitive dependency should exist in a relation R.

Transition Dependency

If a non-prime attribute is determined by another non-prime attribute such dependency is known as transitive dependency. According to 3NF, a relation should not contain transitive dependency.

Ex: Consider a relation R with attributes ABCD holding the following FDs

AB → C, C → D

Finding candidate key

$(AB)^+$ → ABCD

$(C)^+$ → CD

AB is the key for R

AB → C is a transitive dependency C

→ D is not transitive dependency

R is not in 3NF

Decompose R to be in 3NF

Decompose R in R_1 & R_2

R_1: ABC R_2: CD

R_1(ABC)	AB → C	not transitive dependency
R_2(CD)	C → D	not transitive dependency

→The relation R is in 3NF

4. BCNF (Boyce-Codd Normal Form):

a. A relation R is said to be in BCNF iff:

i. It should be in 1NF, 2NF, 3NF.

ii. Every determinant must be a key for R.

If a FD x → y where y → x, x must be a key for R.

b. BCNF is an extension to 3NF so sometimes it is called as 3.5 normal form.

c. BCNF is a strict version of 3NF.

Ex: Consider a relation R with R(ABC) & FDs

AB → C

C → B

Finding candidate key $(AB)^+$: ABC	✓	is in BCNF
$(C)^+$: CD	×	not in BCNF

R is not in BCNF

Now decompose R to be in BCNF

R to R_1 & R_2

R_1(ABC): AB → C AB is a key

R_2(CB): C → B C is a key

∴R is in BCNF

5. Multi Valued Dependencies (MVD) & Forth Normal Form (4NF):

A relation R is said to be in 4NF iff:

a. It should be in 1NF, 2NF, 3NF & BCNF.

b. No multi valued dependencies should be existing in R.

4NF is based on the concept known as Multi Valued Dependencies (MVD).

MVD

MVD is the dependency in which an attribute represents multi-valued facts about another. To determine MVD, a relation should contain 3 or more attributes and the attributes are independent to each other.

Ex: Consider a relation given

Person	Mobile	Food
p1	m1 m2	f1 f2
p2	m3	f3

Convert to 1NF ∴

Person	Mobile	Food
p1	m1	f1
p1	m1	f2
p1	m2	f1
p1	m2	f2
p2	m3	f3

MVD to R

x →→ y y

→→ z

4NF: A relation is said to be in 4NF if it doesn't contain multi valued dependencies

Ex: Consider a relation given

Name	Computer	Language
Aman	Apple Window	Hindi English
Anil	Linux	Hindi English

Convert to 1NF ∴

Name	Computer	Language
Aman	Apple	Hindi
Aman	Apple	English
Aman	Window	Hindi
Aman	Window	English
Anil	Linux	Hindi
Anil	Linux	English

The given Relation R has 2 - MVD

MVD: 1) Name →→ Computer

2) Name →→ Language

∴R is not in 4NF

Decompose R into 2 relations

Name	Computer
Aman	Apple
Aman	Window
Anil	Linux

Name	Language
Aman	Hindi
Aman	English
Anil	Hindi
Anil	English

* Composite key (Name, Computer)
* Composite key (Name, Language)

∴R is in 4NF.

Ex: Consider R(ABCD) which hold FDs

A → BC
CD → E
B → DE
→ A

Find candidate key of R

Sol: $(A)^+$: ABCDE

$(CD)^+$: CDEAB

$(B)^+$: BDEAC

$(E)^+$: EABCD

⍰All are candidate key.

6. Join Dependencies & Fifth Normal Form (5NF):

A relation R is said to be in fifth normal form iff

a. It should be in 1NF, 2NF, 3NF, BCNF & 4NF.

b. It should not be further decomposed or non-join dependency.

5NF is also called as Projection Join NF (PJNF). It is based on the concept known as join dependencies.

JD: A relation R which can be decomposed into R_1, R_2, R_3 ..., R_i is said to be join dependencies.

⍰ $\mathbf{R_1 (R) \bowtie ⍰ R_2 (R) \bowtie \ldots \bowtie ⍰ R_i (R) = R}$

Consider a relation R given

Name	Skill	Job
Aman	DBA	J1
Anil	Programmer	J2
Rohan	Analyst	J3
Ajay	Tester	J4

R is decomposed into three relation:

R1

Name	Skill
Aman	DBA
Anil	Programmer
Rohan	Analyst
Ajay	Tester

R2

Name	Job
Aman	J1
Anil	J2
Rohan	J3
Ajay	J4

R3

Skill	Job
DBA	J1
Programmer	J2
Analyst	J3
Tester	J4

JD: $R_1 \bowtie R_2 \bowtie R_3 (R) = R$

a. $R_1 \bowtie R_2$

Name	Skill	Job
Aman	DBA	J1
Anil	Programmer	J2
Rohan	Analyst	J3
Ajay	Tester	J4

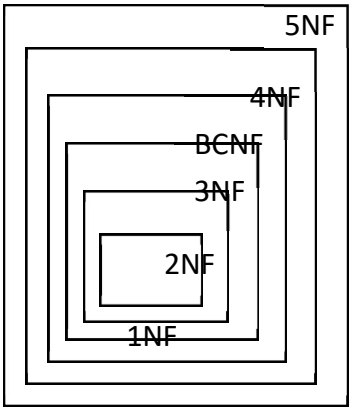
b. $R_1 \bowtie R_2 \bowtie R_3$

Name	Skill	Job
Aman	DBA	J1
Anil	Programmer	J2
Rohan	Analyst	J3
Ajay	Tester	J4

The relation R is a lossless decomposition. Hence, the JD is satisfied.

⍰The relation R is not in 5NF

* **Summary**



1NF: Only atomic values

2NF: No partial dependency
3NF: No transitive dependency
BCNF: Every determinant must be a key
4NF: No Multi Valued Dependencies
5NF: R should be lossless decomposition

- Decomposition

One of the possible solutions for redundancy is decomposition. Decomposition is a process of converting a larger relation into smaller relation. Whenever a relation is decomposed into smaller relations, care must be taken otherwise it leads to 2 problems:

1. Lossless join decomposition 2. Dependency preventing decomposition

- Problems related to decomposition

1. Lossless join decomposition 2. Dependency preserving decomposition

1. Lossless join decomposition

The lossless join decomposition property says that when we have a relation 'R' decomposed into several relations (R_1, R_2, ..., R_i), then we can recover the original relation 'R' by joining the decomposed relations together.

$R = \{R_1, R_2, R_3, ..., R_i)$

$R_1 \bowtie R_2 \bowtie ... \bowtie R_i = R$

v. Rules to check whether a relation is lossless or not:

a. $attr(R_1) \cup attr(R_2) \cup ... \cup attr(R_i) = attr(R)$
The first rule says that we can recover all the attributes of R from the union of attributes of decomposed relations.

b. $attr(R_1) \cap attr(R_2) \neq o$
This rule says that the intersection of attributes of R_1 & attributes of R_2 should not be empty.

c. $attr(R1) \cap attr(R_2) = attr(R_1)$ or $attr(R_2)$
The intersection of attributes of R_1 & attributes of R_2 should result either attributes of R_1 or attributes of R_2

d. Finally, we have to find the closure of intersection attributes.

Ex 1: Consider a relation R (ABCDE) which is decomposed into R_1(ABC) & R_2(ADE) having the following FDs over R

R: A → BC CD → E B → D E → A

Find whether this decomposition of R is lossless or not

Sol:

i. $(R_1 \cup R_2) = R$

(ABC) ∪ (ADE) = (ABCDE) ⊇ R

ii. $R_1 \cap R_2 \neq o$

(ABC) ∩ (ADE) = A ≠ o

iii. $R_1 \cap R_2 = R_1$, $R_1 \cap R_2 = R_2$

(ABC) ∩ (ADE) = A ⊉ R_1, R_2

iv. Closure of A

$(A)^+$ = (ABCDE) ⊇ R

⊇ The Relation is lossless

2. Dependency preserving decomposition

A relation R is decomposed into R1, R2, ..., Ri with the set of FDs given then R is said to be having dependency preserving iff

$$(F)+ = \{F_1 \cup F_2 \cup F_3 \cup \ldots \cup F_n\}$$

Ex1: Consider a relation R with attributes ABC is decomposed into 2 relation AB, BC holding following FDs

A ⊇ B

B ⊇ C

C ⊇ A

Find whether R is dependency preserving or not

AB	BC
A ⊇ BB ⊇ A	B ⊇ C C ⊇ B

$$(F)^{+} = \{F_1 \cup F_2 \cup F_3\}^{+}$$

$$\left\{\begin{matrix} A \to B \\ B \to A \end{matrix}\right\} \cup \left\{\begin{matrix} B \to C \\ C \to B \end{matrix}\right\} = \left\{\begin{matrix} A \to B \\ B \to C \\ C \to A \end{matrix}\right\}$$

$$\left.\begin{matrix} C \to B \\ B \to A \end{matrix}\right\} C \to A \qquad \left.\begin{matrix} B \to C \\ C \to A \end{matrix}\right\} B \to A \qquad \left.\begin{matrix} C \to A \\ A \to B \end{matrix}\right\} C \to B$$

R is dependency preserving

UNIT 4

- UNIT - IV Transaction Concept, Transaction State, Implementation of Atomicity and Durability, Concurrent Executions, Serializability, Recoverability, Implementation of Isolation, Testing for serializability, Lock Based Protocols, Timestamp Based Protocols, Validation- Based Protocols, Multiple Granularity, Recovery and Atomicity, Log–Based Recovery, Recovery with Concurrent Transactions.

-

- Transaction concept

A transaction is an execution of user program that is seen by Database Management System as a series of operations.

(OR)

A transaction is a logical unit of work done by the user.

- Transaction operations

The common operations that can be performed on a transaction include:

1. Read
2. Write

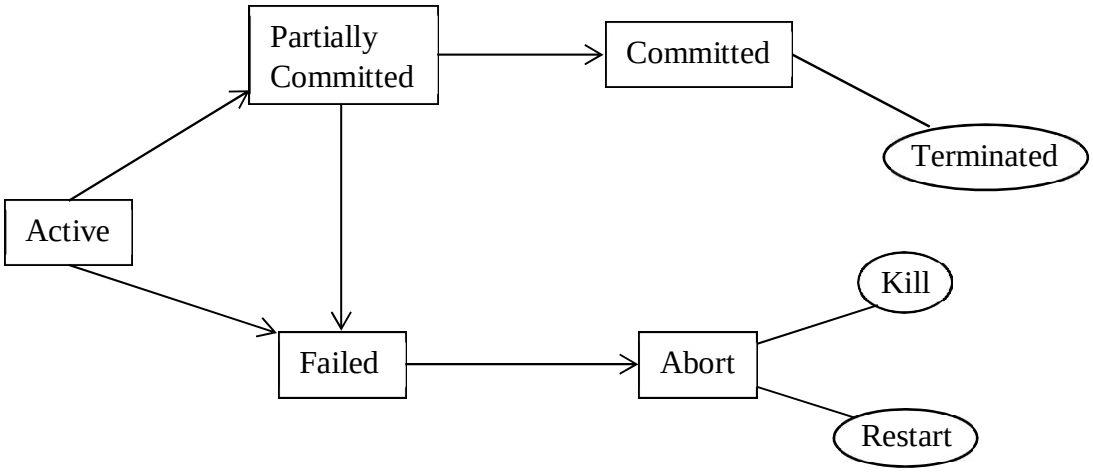
Ex: Consider a bank transaction where the operations debit credit is collectively known as transactions.

- Transaction properties

To maintain consistency of a database, every transaction must satisfy a set of properties known as "ACID" properties. **ACID** - Atomicity Isolation Consistency Durability

1. **Atomicity**
   It ensures that either the transaction operations are executed successfully or not. That means incomplete transactions are not allowed. Atomicity is taken by "Transaction Management Component".
2. **Consistency**
   A transaction that is performed on a consistent state of the database should result to another consistent of database. It is taken care by application layer.
3. **Isolation**
   The transactions executing in the system must have logical isolations. This property ensures thatthe transactions that are executing in parallel will not interfere with each other. It is taken care by concurrency control component.
4. **Durability**
   It ensures that when a transaction is performed on a database the modification done on the database should remain persistent even though the system failure occurs. Durability is taken careby "Recovery Management Concept".

- Transaction state



A transaction executing in the system will enter into different states during lifetime.

1. Active state 2. Partially committed state 3. Failed state 4. Committed state
5. Abort State 6. Terminated state

1. Active state

A transaction is in active state while it is in execution.

2. Partially committed state

A transaction is in partially committed state when it is executing its last statement even after the execution, the transaction may either be committed state or failed state.

3. Failed state

A transaction is in failed state when it no longer continues its normal execution.

4. Committed state

A transaction is in committed state when it completes its execution successfully.

5. Abort state

When a transaction fails all its operation, it must be role back & the transaction enters into abort state. An aborted transaction will be either killed or restarted.

6. Terminated

It is the end of the transaction.

- Schedule

A schedule is a list of operations from a set of transactions.

- Types of schedules

1. Serial schedule 2. Non-serial schedule

Consider 2 transactions T_1, T_2 where transaction T_1 will transfer the amount of 50 from A to B & transaction T_2 transfers the amount of 500 from A to B.

Initial amount of A = 1000

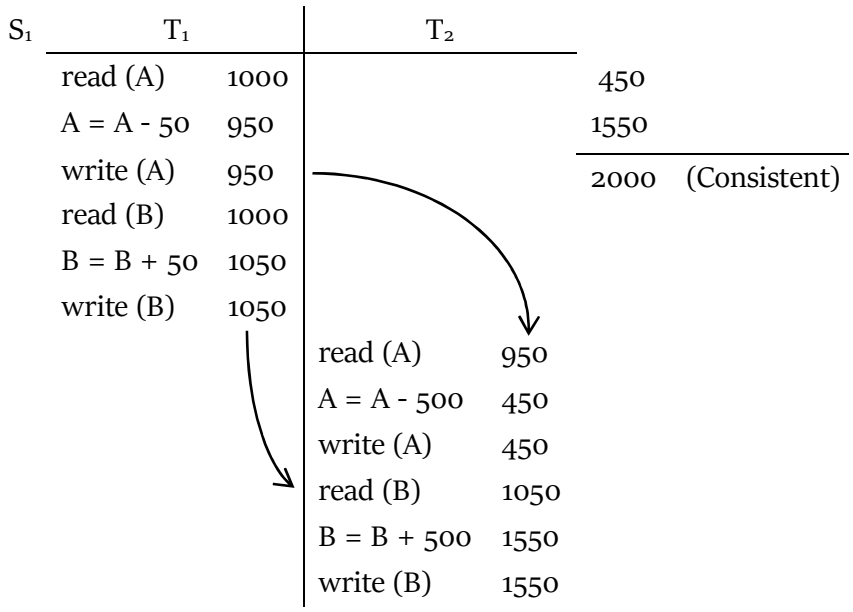
Initial amount of B = 1000

T_1:	read (A)		T2:	read (A)	
	A = A - 50	Debit		A = A - 500	Debit
	write (A)			write (A)	
T2:	read (B)		T2:	read (B)	
	B = B + 50	Credit		B = B + 500	Credit
	write (B)			write (B)	

1. Serial schedule

It is a schedule in which the transaction operations are executing in a serial fashion i.e. T_1 → T_2 or T_2 → T_1.

Ex: Consider a schedule S_1 with 2 transaction T_1 & T_2

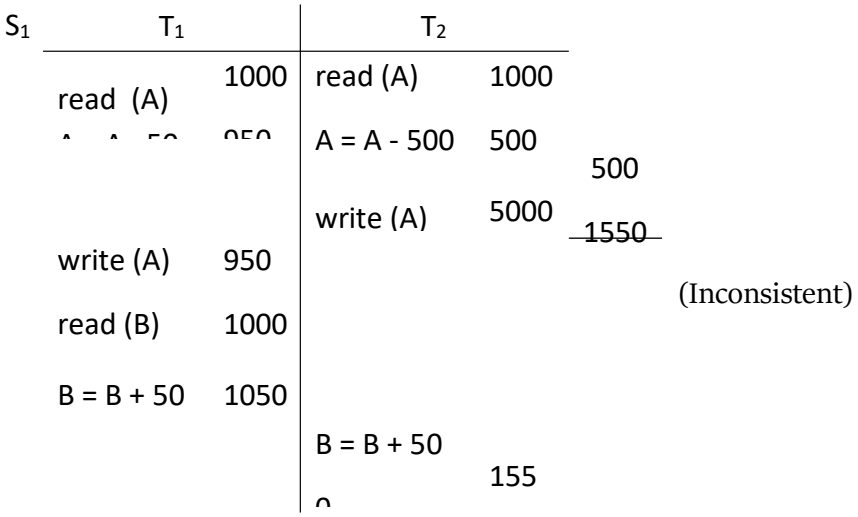


The above schedule S_1 will give consistent result. Hence, it is a serial schedule.

2. Non - Serial schedule (Concurrent or Interleaved)

It is a schedule in which the transaction operations are executing by interleaving each other.

Consider a schedule S_2 as given



The above schedule S_2 is inconsistent because the transactions are operating concurrently.

Note: Non-serial schedule may give consistent or inconsistent result.

- Serializability

A schedule S is said to be serializable if the interleaved execution of transactions is similar to that of some serial schedule.

- Types of serializability

There are 2 types of serializability:

1. Conflict Serializability 2. View Serializability

1. Conflict Serializability

Two schedules S_1, S_2 are said to be conflict serializability if it is conflict equivalent to some serial schedule.

* **Conflict equivalent**

Two schedules S_1, S_2 are said to be conflict equivalent if the conflict operations are executing in the same order.

* **To check conflict equivalent of schedules:**
  a. There must be 2 or more different transactions.
  b. The transactions will work on the same data item.
  c. Atleast one should be 'write' operation.

Consider 2 schedules S_1, S_2 given

S_1	T_1	T_2
	R(A)	
	W(A)	
		R(A)
		W(A)
	R(B)	

S_2	T_1	T_2
	R(A)	
	W(A)	
	R(B)	
	W(B)	
		R(A)

Checking conflict equivalence on S_1, S_2

S_1	T_1		T_2
	R(A)		W(A)
	W(A)		R(A)
	R(A)		W(A)
	R(B)		W(B)
	W(B)		R(B)
	W(B)		W(B)

S_2	T_1		T_2
	R(A)		W(B)
	W(A)		R(A)
	W(A)		W(A)
	R(B)		W(B)
	W(B)		R(B)
	W(B)		W(B)

Since, S_1 & S_2 are conflict equivalent hence, S_1 S_2

2. View Serializability

Two schedules S_1, S_2 are said to be view serializable if it is view equivalent to some serial schedule.

* **View equivalent**

  Two schedules S_1, S_2 are said to be view equivalent if the conflicting operations are executing in the same order.

* **Conditions to check view equivalent**
  a. Check the transaction which reads initial value of data items.
  b. Check the transaction which writes final value of data item.
  c. Check for write read conflicts.

Consider 2 schedules S_1, S_2 given:

S_1	T_1	T_2
	R(A)	
	A = A + 10	
	W(A)	
	R(B)	
	B = B + 20	
	W(B)	
		R(A)
		A = A + 30
		W(A)

S_2	T_1	T_2
	R(A)	
	A = A + 10	
	W(A)	
		R(A)
		A = A + 20
		W(A)
	R(B)	
	B = B + 30	
	W(B)	

Checking of view equivalent

	S_1		S_2		
A	T_1	T_2	T_1	T_2	
B					
W ? R	T_1	T_2	T_1	T_2	
	W(A) ?	R(A)	W(A) ?		R(A)
					R(B)

Since S_1 & S_2 are view equivalent hence, S_1 & S_2 are view serializable.

- Testing of serializability

In order to test serializability (Conflict serializability) of a given schedule, we use precedence graph. A precedence graph is a directed graph where G = (V, E).

i. Algorithm for creating a precedence graph:

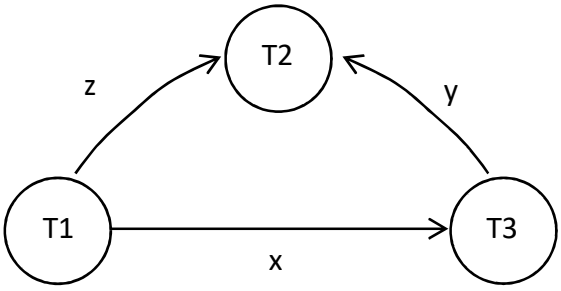
Step 1: Create a node for each transaction.

Step 2: Draw a directed edge from T_i to T_j (T_i ? T_j) for the following cases

i. $T_i(w)$? $T_j(r)$
ii. $T_i(r)$? $T_j(w)$
iii. $T_i(w)$? $T_j(w)$

Ex1: Consider a given schedule S_1. Check whether S_1 is conflict serializable or not

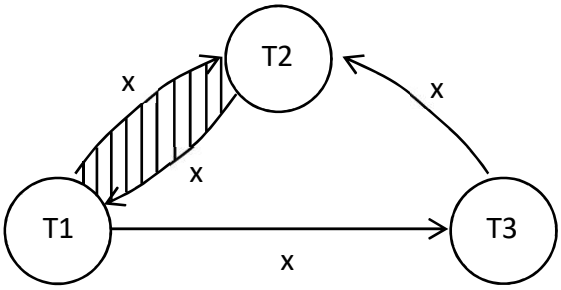
S_1	T_1	T_2	T_3
	r(x)		
		r(z)	
	r(z)		
			r(x)
			r(y)
			w(x)



No cycle in graph hence, S_1 is conflict serializable

Ex2:

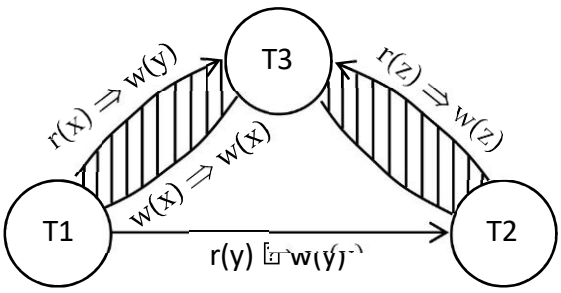
S_2	T_1	T_2	T_3
	r(x)		
			r(x)
			w(x)



It has cycle in graph so S_1 is not conflict serializable

Ex3:

S_3	T_1	T_2	T_3
	r(x)		
			r(z)
			w(z)
		r(y)	
	r(y)		
		w(y)	



There is a cycle in the graph hence, it is not conflict serializable.

- Recoverability

A transaction may not execute completely due to failure such as hardware or software failure. In that case we have to roll back the failed transaction but some other transactions are using the values written by failed transaction. So, we have to roll back those transactions as well.

Ex: Consider a given schedule S_1

S_1	T_1	T_2
	r(A)	
	A = A + 50	
		r(A)
		A = A + 100
		w(A)
	Failure point	
	commit	

In the above schedule, transaction T_1 fails due to some reason so we roll back T_1. In such case, T_2 must also be rolled back because T_2 reads the value written by T_1 but in the above schedule, T_2 commits before T_1. Hence it cannot be rolled back. This phenomenon is known as irrecoverable schedules.

* **Recoverable schedule**

A schedule is recoverable where each transaction commits only after all transaction from which it has read as committed.

Ex:

S_2	T_1	T_2
	r(A)	
	A = A + 50	
		r(A)
		A = A + 100
	Commit	
		Commit

In the above schedule, transaction T_2 commits after T_1 commits. If T_1 fails then we can roll back T_1 as well as T_2. Therefore, the schedule S_2 is said to be recoverable schedule.

- Cascading roll back

Consider a schedule S_3 given

S_3	T_1	T_2	T_3
	r(A)		
		r(A)	
		w(A)	
			r(A)
	Failure point		

In the above schedule, if transaction T_1 fails then we have to roll back T_1. In such case, T_2 & T_3 must be rolled

back because T_2 depends on T_1 & T_3 depends on T_2. This concept is known as cascading rollback.

- Cascade less schedule

A cascade less schedule is a schedule where for each pair of transaction T_i, T_j such that T_j reads a data item return T_i only after T_i commits.

S_4	T_1	T_2
	r(x)	
		r(y)
	Commit	
		r(x)

Note: Cascade less schedule are always recoverable.

- Concurrency control protocol

Concurrency control is one of the methods which guarantees the consistency of the database even with interleaved execution of transactions. To deal with interleaved execution of transactions, different concurrency protocols are available

1. Lock based protocol 2. Timestamp based protocol 3. Validation based protocol
4. Multiple granularity protocol

1. Lock based protocol

* **Lock:** A lock is a variable that is assigned to a data item which gives the status of data item with respect to the operations allowed on it

* **Types of locks**

  Locks are basically categorized into 2 types.

  a. Shared lock(s) b. Exclusive lock(x)

  a. Shared lock(s)

  It is also known as read only lock. It is denoted with s. Any no of transactions can have a shared lock on a data item.

  b. Exclusive lock(x)

  It is also known as read – write lock. Only 1 transaction at a time can have exclusive lock on a data item.

* **2 Phase Locking (2PL)**

  2PL is a concurrency control protocol available under lock-based protocol. 2PL ensures conflict serializable schedules. 2PL works in 2 different phases

  a. Growing phase b. Shrinking phase

  a. In **growing phase**, a transaction obtains locks but may not release any locks.
  b. In **shrinking phase**, a transaction may release lock but cannot obtain any locks.

* **Types of 2 Phase Locking of lock-based protocols**

  a. Simple 2 phase locking b. Strict 2 phase locking c. Rigorous 2 phase locking

  d. Conservative 2 phase locking

  a. Simple 2 phase locking:

  Consider 2 schedules S_1, S_2 as given

S_1: T_1	T_2	Lock Manager
lock - s(A)		Grant lock - S(A)
r(A)		
unlock(A)		Grant lock - x(B)
lock - x(B)		
r(b)		
w(b)		
	lock - s(A)	Grant lock - S(A)
	r(A)	
	unlock(A)	Grant lock - x(B)
	lock - x(B)	
	r(b)	
	w(b)	

Schedule $S_1(T_1, T_2)$ is not in 2PL. It does not obey the properties of 2PL because in T_1 lock(x(B)) is executed after unlock(A) & similar in T_2.

S_2: T_1	T_2	Lock Manager
lock - s(A)		Grant lock - S(A)
r(A)		
		Grant lock - x(B)
lock - x(B)		
unlock(A)		
r(b)		
w(b)		
	lock - s(A)	Grant lock - S(A)
	r(A)	
		Grant lock - x(B)
	lock - x(B)	
	unlock(A)	
	r(b)	
	w(b)	

Schedule $S_2(T_1, T_2)$ obey the properties of 2PL so S_2 is in 2 Phase locking

- **Advantages**

  It ensures conflict serializability.

- **Disadvantages**

  i. It does not ensure freedom from deadlocks

  ii. There is a possibility of cascading rollback to occur

b. Strict 2 phase locking

It is compatible with 2PL. It does not release x lock (exclusive lock) until the transaction commits. It guarantees conflict serializability.

Consider a schedule S_3 as given:

S_3:	T_1	T_2
	lock - x(P)	
	r(P)	
	w(P)	
	lock - x(Q)	
	r(Q)	
		lock - x(P)
		r(P)
		w(P)
		lock - x(Q)
		r(Q)

The $S_3(T_1, T_2)$ obeys strict 2PL

c. Rigorous 2 phase locking

It is compatible with 2PL. The shared locks & exclusive locks must be released only after transaction commit or abort. It is a strict version of strict 2PL.

Consider a schedule S_4 as given:

S_4:	T_1	T_2
	lock - s(P)	
	r(P)	
	lock - x(Q)	
	r(Q)	
	w(Q)	
		lock - s(P)
		r(P)
		lock - x(Q)
		r(Q)
		r(Q)

The $S_4(T_1, T_2)$ obeys rigorous 2PL.

Note: Strict 2PL, rigorous 2PL does not ensure freedom from deadlock.

d. Conservative 2 phase locking

The conservative 2PL says that a transaction should obtain all the required locks (shared or exclusive) before it starts its execution & release all the locks after it commits. It ensures freedom from deadlock.

Consider a schedule S_5 as given:

S_5: T_1	T_2
lock - s(A)	
lock - x(B)	
r(A)	
r(B)	
	lock - s(A)
	lock - x(B)
	r(A)
	r(B)

The $S_5(T_1, T_2)$ obeys conservative 2PL.

* **Lock conversions**

Lock conversion is a concept in which we can either upgrade or degrade the locks assigned to a transaction. The lock conversion is necessary to overcome deadlock situation occurred in a schedule. **Ex: Consider a transaction T_1 holding a shared lock on 'A'.**

T_1: lock - s(A)

If T_1 wants to perform 'write' operation on data item 'A', then it should request for exclusive lock on A. This will be done by upgrading lock

T_1: lock - s(A)

- upgrading

lock - x(A)

Consider T_2: lock - x(B). Suppose T_2 does not require the x lock on B anymore, then we degrade the lock to shared lock.

T_2: lock - x(B)

- degrading

lock - s(B)

* **Graph-based protocol (Tree locking protocol)**

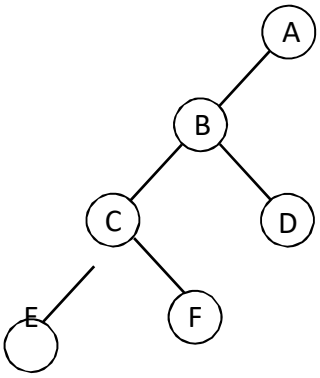
a. It is a simple locking protocol i.e. not 2 phase locking.
b. This protocol is basically designed for exclusive locks.
c. This protocol requires a prior knowledge regarding the order in which data items can be accessed.
d. This knowledge can be acquired by using partial ordering on the given data items.
e. This protocol ensures conflict serializability.

* **Working of graph-based protocol**

In tree locking protocol, every transaction should consider the following conditions

a. The transaction T_i can lock any data item initially.
b. The T_i can lock a data item 'X' only if it has a lock on parent of X.
c. Once a transaction T_i lock and unlock a data item, then it cannot request for a lock on the same data item.

Ex: Consider the given set of data item in a tree like structure



Consider a schedule $S_1(T_1, T_2, T_3)$

S_1:	T_1	T_2	T_3
	lock - x(B)		
	r(B)		
		lock - x(C)	
		r(C)	
		lock - x(F)	
		r(F)	
	lock - x(D)		
	r(D)		
	w(D)		
	lock - x(C)		
	r(C)		
	w(C)		
			lock - x(B)
			r(B)
			w(B)
			lock - x(D)
		unlock - F	
	lock - x(E)		
	r(E)		
	w(E)		
			unlock - D
	unlock - E		

2. Timestamp based protocol

The timestamp ordering is a method that determine the serializability if different transaction in a schedule.

* **Timestamp**

A timestamp is an identifier that specifies the starting time of transaction & it is generated by database system. Each transaction will have a unique timestamp. It is denoted by $T_S(T_i)$.

The time stamps are generated by using 2 methods

a. System clock b. Logical counter

a. System clock

When a transaction enters into a system, it is assigned with a timestamp value equal to system clock.

b. Logical counter

Each transaction is assigned with a counter value & it is incremented for every new transaction that enters into the system

* **Types of timestamps**

a. Read Timestamp (RTS) b. Write Timestamp (WTS)

a. RTS

It indicates the highest value of timestamp generated by the transaction for reading a data item.

b. WTS

It indicates the highest value of timestamp generated by the transaction for writing a data item.

In timestamp ordering protocol, the timestamp of an old transaction is always less than the timestamp of new transaction i.e. $T_S(T_i) < T_S(T_j)$

where T_i - Old transaction

T_j – New transaction

* **Working of timestamp ordering protocol**

I: When a transaction (T_i) issue a read operation on X

i. $T_S(T_i) < WTS(X)$ //read is rejected

ii. $T_S(T_i) \geq WTS(X)$ //read is executed & update WTS value with $T_S(T_i)$.

II: When a transaction (T_i) issue a write operation on X

i. $T_S(T_i) < RTS(X)$ //write is rejected

ii. $T_S(T_i) < WTS(X)$ //write is rejected

iii. $T_S(T_i) \geq WTS(X)$ //write is executed & WTS is updated.

S_1:

T_1	T_2
r(A)	
	r(A)
	A = A + 10
r(B)	
	r(B)
show (A + B)	
	show (A + B)

Schedule S_1 obeys timestamp ordering by $T_S(T_1) < T_S(T_2)$.

- Validation based protocol:

1. It is an optimistic concurrency control protocol.
2. It is based on timestamp ordering technique. Each transaction is executed in 3 phases

a. Read - Read b. Validation - Validation c. Write - Write

a. Read phase (T_i)

In **Read - Read** transaction T_i, x is executed by reading various data items and storing them in local variable.

b. Validation phase (T_i)

In this phase, the validation test conducted for transaction T_i against serializability order.

c. Write phase (T_i)

In this phase, the transaction T_i will update the data items in the database.

Time stamps are associated with 3 – phases

1. Start (T_i) 2. Validate (T_i) 3. Finish (T_i)

1. Start (T_i)

It indicates the time when T_i started its execution.

2. Validate (T_i)

It indicates the time when T_i started its validation test.

3. Finish (T_i)

It indicated the time when it finishes the write phase.

- Working with timestamp-based protocol

- Validation test

Consider 2 transaction T_1, T_2 is executing in the system with timestamp ordering $T_S(T_1) < T_S(T_2)$ then

Validation test for T_1, T_2:

1. finish (T_1) < start (T_2)
2. writes of T_1 & T_2 do not overlap

Ex: Consider a schedule S_1 with T_1, T_2

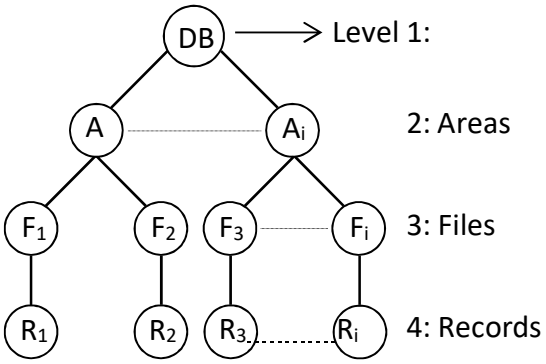
S_1	T_1	T_2
	R(A)	
	W(A)	
	R(B)	
		R(A)
		R(B)
		<validate>

 S_1 satisfy validate test

- Multiple granularity

1. In earlier concurrency control protocol, locking can be applied to a single data item. Sometimes we need to lock a collection of data items by a transaction i.e. possible by using granularity process. Granularity indicates the size of data item allowed to lock.
2. Multiple granularity can be define as a hierarchy that will breakup the database into different blocks which can be locked by the transaction.

Consider a granularity hierarchy as given below:



- Working of granularity hierarchy

1. Each node can be locked individually.
2. When a node is locked using either shared or exclusive lock explicitly by a transaction, then all the descends of that node get the same lock explicitly.
3. When a transaction wants to lock the entire database, which effects the concurrency of the system, it requires a new locking mechanism known as intension lock nodes which are

   a. IS (Intention Shared) b. IX (Intention Exclusion)

Compatibility matrix:

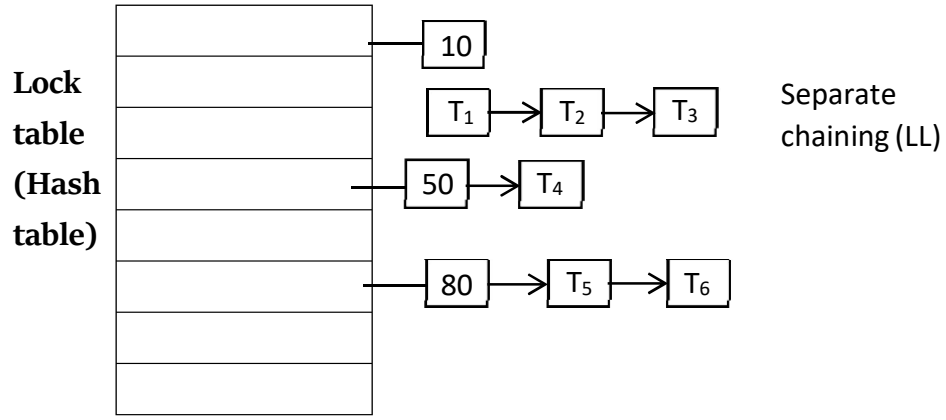
	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	×
IX	✓	✓	×	×	×
S	✓	✓	✓	×	×
SIX	✓	×	×	×	×
X	✓	×	×	×	×

- Implementation of locking (processing lock and unlock requests by lock manager)

A lock manager is implemented as a process that will receive lock request, messages from transactions and give the response to the transaction.

The lock manager after receiving the lock request from a transaction process and grant the lock of the data item if it is available free. To process the lock request, the lock manager uses the following data structure.

Data structure used by lock manager



H(k) = h(k) ≠ 10

The lock table maintained the information about the data items that are locked and unlocked by transactions. It uses a separate chaining technique to maintain a linked list of data items for each entry in the lock table. The transactions which are waiting for the locks are added to the linked list once a transaction released a lock, it is granted to one of the transactions in linked list.

- Recovering techniques

- Failure with loss of non-volatile storage:

The information present in a volatile storage gets lost whenever a system crash occurs. But the lossof information in a non-volatile storage is very rear to avoid such failure, some certain techniques need to be considered.

One of the techniques is dump. In this technique, the entire database is dump to a stable storage at regular intervals of time. When a system crashes, in order to bring the database back to consistent state, weuse the recent dump to restore.

* Dump process:

1. During dump process, no transaction should be processed.
2. All the log records in the memory must be stored to a stable storage.
3. The entire database is copied to stable storage.

Drawbacks of dump process

The dump process is considered as an expensive task due to following reasons.

1. The huge amount of data transfer is needed.
2. No transaction is in process during dump, hence CPU cycle is waste.

- Remote backup system

The traditional transaction processing system is more suspected to failure due to natural disaster. Hence there is a need for designing a system which will continue its processing even if the system fails due to natural disaster. Such a system is known as remote backup system. The main goal of remote backup systemis to provide high degree of availability.

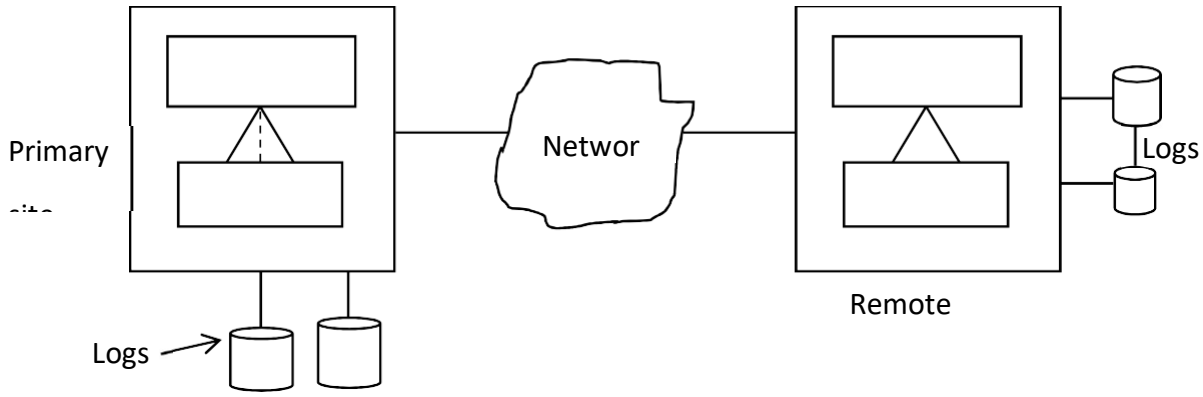


Fig: Remote backup system

- Working of remote backup system

1. When a primary site is updated, this updation should be reflected in the remote site. This can be doneby synchronization between primary site and remote site.
2. When primary site fails, the remote site immediately takes the responsibility of transaction processing.

- **Design issues of remote backup system**
  1. **Failure detection 2. Control of transfer 3. Recovery time 4. Time of commit**

-

DATA ON EXTERNAL STORAGE :-

A DBMS stores vast quantities of data, and the data must persist across program executions Therefore, data is stored on external storage devices such as disks and tapes, and fetched into main memory as needed for processing. The unit of information read from ~~to~~ written to disk is a page. The size of a page is a DBMS parameter, and typical values are 4KB or 8KB.

The cost of page I/O (input from disk to main memory and output from memory to disk) dominates the cost of typical database operations, and database systems are carefully optimized to minimize this cost.

Some of important points are:

- Disks are the important external storage devices. They allow us to retrieve any page at a (more/less) fixed cost per page. However, if we read several pages in the order that they are stored physically, the cost can be much less than the cost of reading the same pages in a random order.
- Tapes are sequential access devices and force us to read data one page after the other. They are mostly used to archive data that is not needed as an

regular basis.

- Each record in a file has a unique identifier called a record id, or rid for short. An rid has the property that we can identify the disk address of the page containing record using the rid.

Data is read into memory for processing, and written to disk for persistent storage, by a layer of software called the buffer manager. The buffer manager fetches the page from disk if it is not already in memory.

Space on disk is managed by disk space manager. The disk space manager keeps track of the pages in use by the file layer; if a page is freed by the file layer; the space manager tracks this, and reuses the space if file layer requests a new page later on.

FILE ORGANISATION AND INDEXING:-

The file of records is an important abstraction in a DBMS, and is implemented by file and access methods layer of the code. A file can be created, destroyed, and have records inserted into and deleted from it. It also supports scans; a scan operation allows us to step through all the records in the file one at a time. A relation is typically stored as file of records.

The simplest file structure is an unordered file, or heap file. Records in a heap file are stored in random order across the pages of file.

An index is a data structure that organizes data records on disk to optimize certain kinds of retrieval operations. An index allows us to efficiently retrieve all records that satisfy search conditions on search key fields of the index.

We use the term data entry to refer to records stored in an index file. A data entry with search key value k, denoted as k*, contains enough information to locate (one or more) data records with search key value k.

There are three main alternatives for what to store as a data entry in an index:

1. A data entry k* is an actual data (with search key value k).
2. A data entry is a <k, rid> pair, where rid is the record id of a data record with search key value k.
3. A data entry is a <k, rid-list> pair, where rid-list is a list of records ids of data records with search key value k.

A indexed file organization can be used instead of, for example, a sorted file or an unordered file of records.

CLUSTERED INDEXES:-

When a file is organized so that the ordering of data records is the same as or close to the ordering of data entries in some index, we say that the index is clustered; otherwise it clustered is an unclustered index. An index that uses Alternative (1) is clustered, by definition. We sometimes refer to an index using Alternative(1) as a clustered file, because the data entries are actual data records, and index is therefore a file of data records.

The cost of using an index to answer a range search query vary tremendously based on whether the index is clustered.

PRIMARY AND SECONDARY INDEXES:-

An index on a set of fields that includes the primary key is called a primary index; other indexes are called secondary indexes.

Two entries are said to be duplicates if they have the same value for search key field associated with the index. A primary index is guaranteed not to contain duplicates, but an index on other fields can contain duplicates. In general, a secondary index contain duplicates. If we know that no duplicates exist, that is, we know

that the search key contains some candidate key, we call the index a unique index.

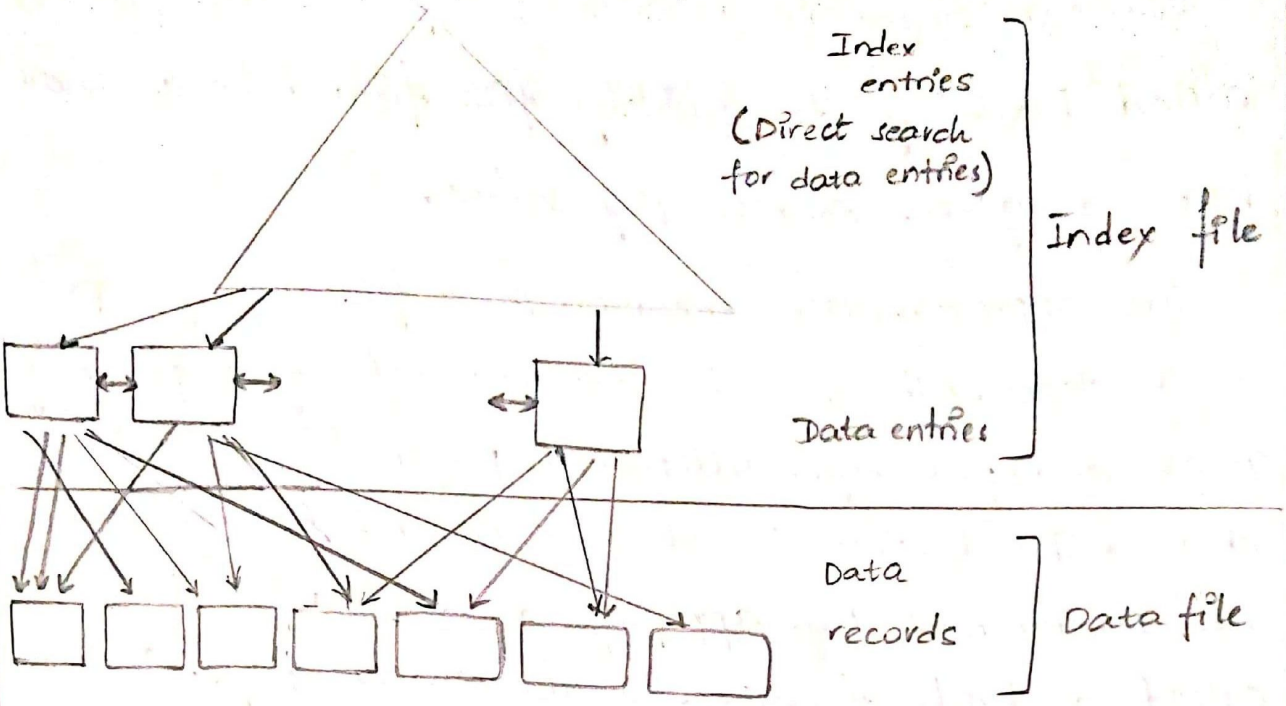


Fig: Unclustered Index using Alternative (2)

INDEX DATA STRUCTURES:-

One way to organize data entries is to hash data entries on the search key. Another way to organize data entries is to build a tree-like data structure that directs a search for data entries.

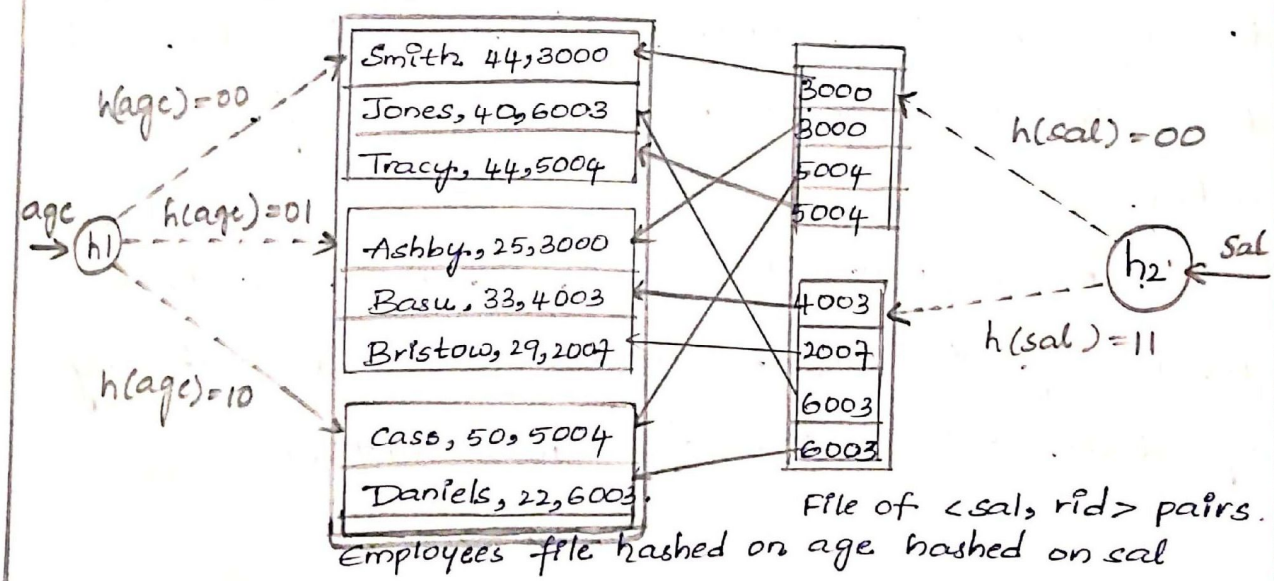
We note that the choice of hash or tree indexing techniques can be combined with any of three alternatives for data entries.

HASH TABLE INDEXING:-

We can organize records using a technique called hashing to quickly ~~file~~ find records that have a given search key value.

In approach, the records in a file are grouped in buckets, where a bucket consists of a primary page and, possibly, additional pages linked in a chain. The bucket to which a record belongs can be determined by applying a special function, called a hash function, to the search key. Given a bucket number, a hash-based index structure allows us to retrieve the primary page for the bucket in one or two disk I/Os.

On inserts, the record is inserted into the appropriate bucket, with 'overflow' pages allocated as necessary.



Note that the key for an index can be any sequence of one or more fields, and it need not uniquely identify records.

TREE-BASED INDEXING:-

An alternative to hash-based indexing is to organize records using a tree-like data structure. The data entries are arranged in sorted order by search key value, and a hierarchical search data structure is maintained that directs searches to correct page of data entries.

The lowest level of tree, called leaf level contains the data entries; in our example, these are employee records.

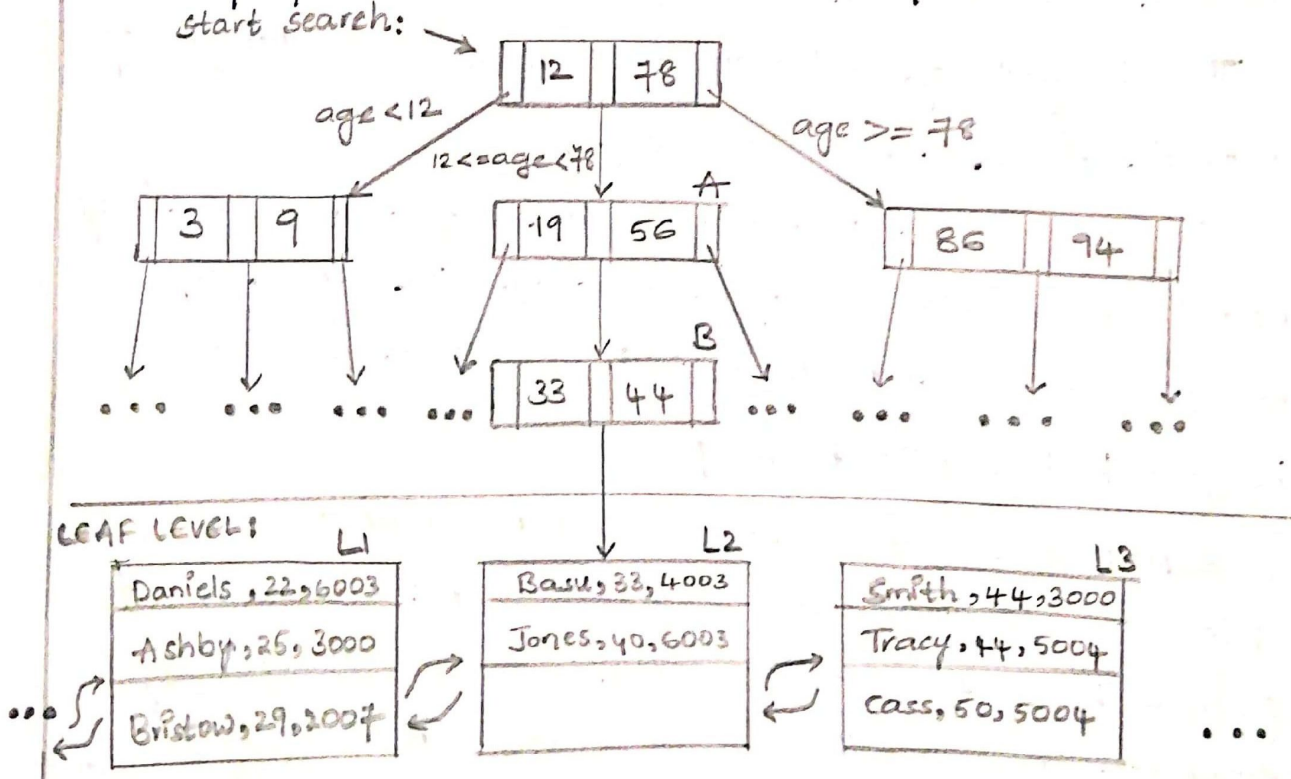


Fig: Tree-Structured Index

This structure allows us to efficiently locate all data entries with search key values in a desired range. All searches begin at topmost node, called the root, and the contents of pages in non-leaf levels direct searches to correct leaf page.

Thus, the no. of disk I/os incurred during a search is equal to length of path from the root to a leaf, plus the no. of leaf pages with qualifying data entries. The B+ tree is an index structure that ensures that all paths from root to leaf in given tree are of same length, that is, the structure is always balanced in height.

The height of a balanced tree is the length of a path from root to leaf.

The average no. of children for a non-leaf node is called the fan-out of the tree. If every non-leaf node has n children, a tree of height h has n^{th} leaf pages.

COMPARISON OF FILE ORGANIZATIONS:-

We now compare the costs of simple operations for several basic file organizations on a collection of employee records. The organizations that we consider are the following:

- File of randomly ordered employee records, or heap file.

- File of employee records sorted on <age, sal>.
- Clustered B+ tree file with search key <age, sal>.
- Heap file with an unclustered B+ tree index on <age, sal>.
- Heap file with an unclustered hash index on <age, sal>

Our goal is to emphasize the importance of the choice of an appropriate file organization, and the above list includes the main alternatives to consider in practice.

The operations we consider are these:

- Scan: Fetch all records in the file. The pages in the file must be fetched from disk into the buffer pool. There is also a CPU overhead per record for locating the record on the page (in pool).
- Search with equality selection: Fetch all records that satisfy an equality selection. Pages that contain qualifying records must be fetched from disk, and qualifying records must be located within retrieved pages.
- Search with range selection: Fetch all records that satisfy a range selection.
- Insert a Record: Insert a given record into file. We must identify the page in the file into which the new record must be inserted, fetch that page from disk, modify it to include the new record,

and then write back the modified page. Depending on file organization, we may have to fetch, modify, and write back other pages as well.

- Delete a Record: Delete a record that is specified using its rid. We must identify the page that contains the record, fetch it from disk, modify it, and write it back. Depending on the file organization, we may have to fetch, modify, and write back other pages as well.

File Type	Scan	Equality Search	Range Search	Insert	Delete.
Heap	BD	0.5 BD	BD	2D	Search + D
Sorted	BD	$D\log_2 B$	$D\log_2 B$ + # matching pages	Search + BD	Search + BD
Clustered	1.5 BD	$D\log_F 1.5B$	$D\log_F 1.5B$ + # matching pages	Search + D	Search + D
Unclustered tree index	BD(R + 0.15)	$D(1 + \log_F 0.15B)$	$D(\log_F 0.15B$ + # matching records)	$D(3 + \log_F 0.15B)$	Search + 2D.
Unclustered hash index	BD (R + 0.125)	2D	BD	4D	Search + 2D

INDEXES AND PERFORMANCE TUNING.

The choice of indexes has a tremendous impact on system performance, and must be made in the context of the expected workload, or typical mix of queries and update operations.

A full discussion of indexes and performance requires an understanding of database query

evaluation and concurrency control.

Clustered Index Organization:

A clustered index is really a file organization for the underlying data records. Data records can be large, and we should avoid replicating them; so there can be at most one clustered index on a given collection of records. We can also build an unclustered index on, say, department, if there is such a field.

Clustered indexes, while less expensive to maintain than a fully stored file, are nonetheless expensive to maintain.

In dealing with the limitation that at most one index can be clustered, it is often useful to consider whether the information in an index's search key is sufficient to answer the query. For example, if we have an index on age, and we want to compute the average age of employees, the DBMS can do this by simply examining the data entries in the index. This is an example of an index-only evaluation. In an index-only evaluation of a query we need not access the data records in files that contain the relations in the query; we can evaluate the query completely through indexes on the file.

Design Examples Illustrating Clustered Indexes:

To illustrate the use of a clustered index on a

range query, consider the following example:

```
SELECT   E.dno.
FROM     Employees E
WHERE    E.age >40.
```

As another example, consider the following refinement of the previous query:

```
SELECT   E.dno, COUNT(*)
FROM     Employees E
WHERE    E.age >10
GROUP BY E.dno.
```

Clustering is also important for an index on a search key that does not include a candidate key that is, an index in which several data entries can have the same key value. To illustrate this point, we present the following query:

```
SELECT   E.dno
FROM     Employees E
WHERE    E.hobby = 'Stamps'.
```

The next query shows how aggregate operations can influence the choice of indexes:

```
SELECT   E.dno, COUNT(*)
FROM     Employees E
GROUP BY E.dno.
```

Index Specification in SQL:1999:

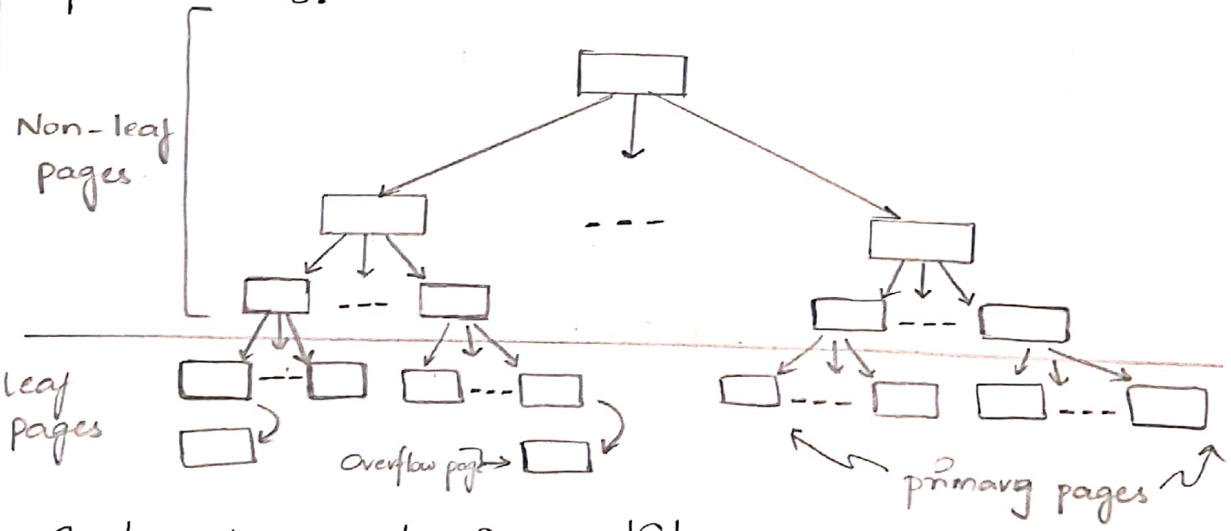
The SQL:1999 standard does not include any

statement for creating or droping index structures.

```
CREATE INDEX IndAgeRating ON Students
        WITH STRUCTURE = BTREE,
             KEY = (age, gpa).
```

INDEXED SEQUENTIAL ACCESS METHODS (ISAM):-

The data entries of the ISAM index are in the leaf pages of the tree and additional overflow pages chained to some leaf page. The ISAM structure is completely static and facilities such low-level optimizations.



Each tree node is a disk page, and all the data resides in leaf pages. When file is created, all leaf pages are allocated sequentially and sorted on search key value. These additional pages are allocated from an overflow area.

Data Pages
Index Pages
Overflow pages

The basic operations of insertion, deletion, and search are all quite straightforward. For an equality selection search, we start all the root node and determine which subtree to search by comparing the value in search field of given record with the key values in the node.

We assume that each leaf page can contain two entries. Chains of overflow pages can easily develop.

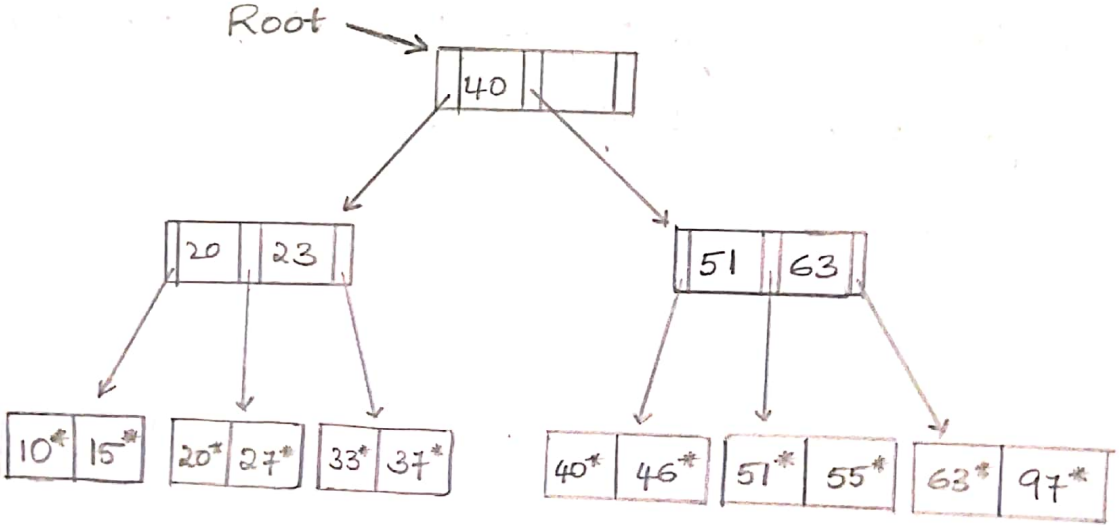


Fig: Sample ISAM Tree.

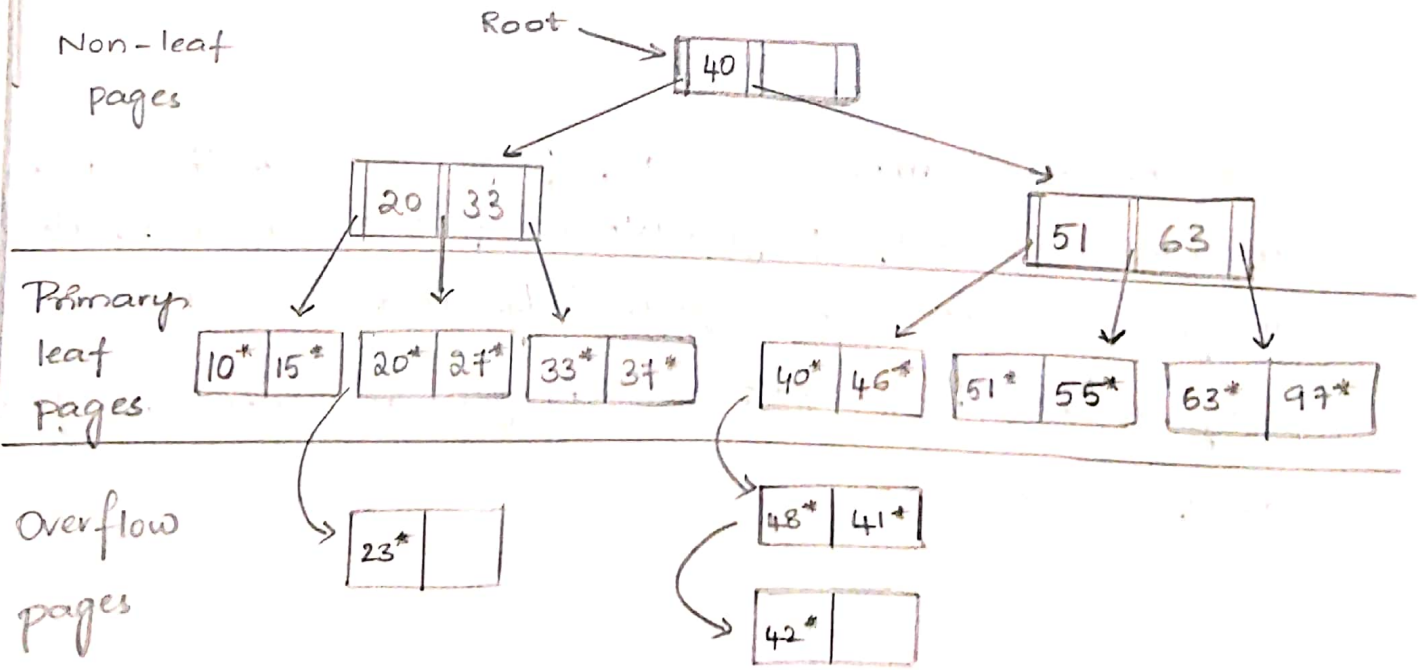


fig: ISAM Tree after Inserts.

Overflow Pages, Locking Considerations:

Once the ISAM file is created, inserts and deletes affect only the contents of leaf pages. A consequence of this design is that long overflow chains could develop if a no. of inserts are made to the same leaf. These chains can significantly affect the time to retrieve a record because the overflow chain has to be searched as well when the search gets to this leaf.

The fact that only leaf pages are modified also has an important advantage with respect to concurrent access. When a page is accessed, it is typically 'locked' by the requestor to ensure that it is not concurrently modified by other users of the page. In the ISAM structure, since we know that index-level pages are never modified, we can safely omit the locking step. Not locking index-level pages is an important advantage of ISAM over dynamic structure like a B+ tree. If data distribution and size are relatively static, which means overflow chains are rare, ISAM might be preferable to B+ trees due to this advantage.

B+ TREES: A DYNAMIC INDEX STRUCTURE:-

A static structure such as the ISAM index

Suffers from the problem that long overflow chains can develop as the file grows, leading to poor performance. This problem motivated the development of more flexible, dynamic structures that adjust gracefully to inserts and deletes. The B+ tree search structure, which is widely used, is a balanced tree in which the internal nodes direct the search and the leaf nodes contain data entries. To retrieve all leaf pages efficiently, we have to link them using page pointers. By organizing them into a doubly linked list, we can easily traverse the sequence of leaf pages (sometimes called the sequence set) in either direction. This structure is illustrated below.

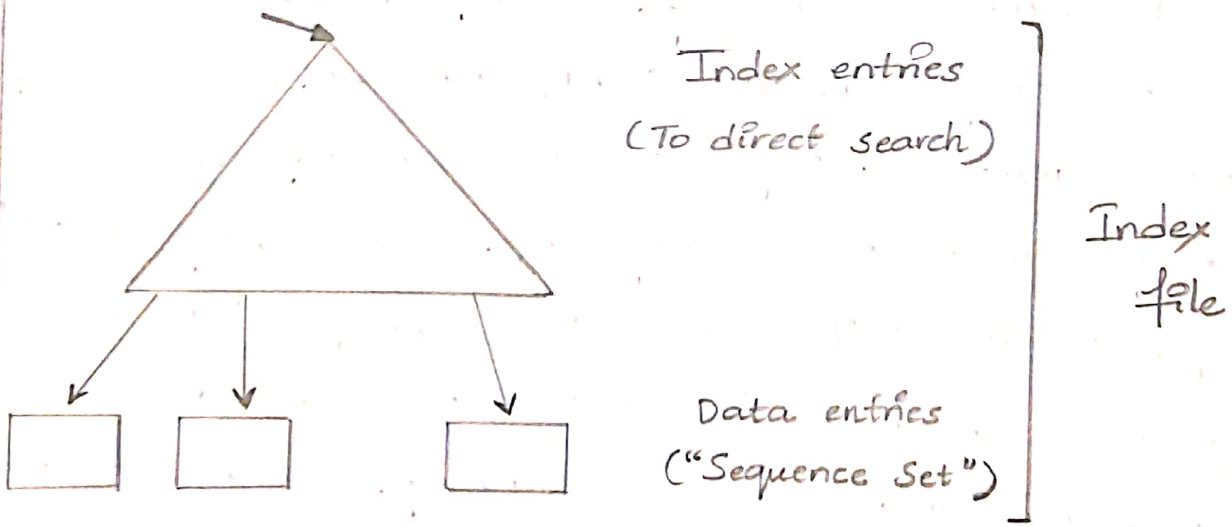


Fig: Structure of a B+ Tree.

The following are some of the main characteristics of a B+ Tree:

- Operations (insert, delete) on the tree keep it balanced.

- A minimum occupancy of 50 percent is guaranteed for each node except the root if deletion algorithm is implemented. However, deletion is often implemented by simply locating the data entry and removing it, without adjusting the tree as needed to guarantee the 50 percent occupancy, because files typically grow rather than shrink.
- Searching for record requires just a traversal from the root to appropriate leaf. We refer to length of a path from the root to a leaf - any leaf, because the tree is balanced - as the height of the tree.

We will study B+ trees in which every node contains m entries, where $d \leq m \leq 2d$. The value d is a parameter of B+ Tree, called the order of the tree, and is a measure of capacity of tree node. The root node is only exception to this requirement on number of entries; for the root, it is simply required that $1 \leq m \leq 2d$.

If a file of records is updated frequently & sorted access is important, maintaining a B+ Tree index with data records stored as data entries is almost always superior to maintaining a stored file. B+ trees are usually also preferable

to ISAM indexing because inserts are handled gracefully without overflow chains. However, if the dataset size and distribution remain fairly static, overflow chains may not be a major problem. As a general rule, however, B+ Trees are likely to perform better than ISAM.

Format of a Node:

Non-leaf nodes with m index entries contain m+1 pointers to children. Pointer P_i points to a subtree in which all key values k are such that $K_i \leq k < K_{i+1}$. For leaf nodes, entries are denoted as k*, as usual. Just as in ISAM, leaf nodes contain data entries. Thus, the leaves form a sequence, which can be used to answer range queries efficiently.

If field being indexed is of fixed length, these index entries will be of fixed length; otherwise, we have variable length records. In either case the B+ tree can itself be viewed as a file of records. If leaf pages do not contain the actual data records, then the B+ tree is indeed a file of records that is distinct from the file that contains the data. If the leaf pages contain data records, then a file contains the B+ tree as well as the data.