



**MALLA REDDY ENGINEERING COLLEGE  
(AUTONOMOUS)**

**Lecture Notes**

**On**

**FDBMS**

**Prepared by,**

**E.SUNIL,**

**ASSISTANT PROFESSOR-CSE  
MREC .**

## **DBMS: Basic Concepts**

1. Introduction
2. Disadvantages of file oriented approach
3. Database
4. Why Database
5. Database Management System(DBMS)
6. Function of DBMS
7. Advantages of DBMS and disadvantage of DBMS
8. Database Basics
9. Three level architecture of DBMS
10. Database users
11. Database language
12. Database structure

### **Introduction:**

In computerized information system data is the basic resource of the organization. So, proper organization and management for data is required for organization to run smoothly. Database management system deals with the knowledge of how data is stored and managed on a computerized information system. In any organization, it requires accurate and reliable data for better decision making, ensuring privacy of data and controlling data efficiently.

The examples include deposit and/or withdrawal from a bank, hotel, airline or railway reservation, purchase items from supermarkets in all cases, a database is accessed.

### **What is data:**

Data is the known facts or figures that have implicit meaning. It can also be defined as it is the representation of facts, concepts or instructions in a formal manner, which is suitable for understanding and processing. Data can be represented in alphabets(A-Z, a-z), in digits(0-9) and using special characters(+, -, #, \$, etc)  
: 25, "ajit" etc.

### **Information:**

Information is the processed data on which decisions and actions are based. Information can be defined as the organized and classified data to provide meaningful values.

Eg: "The age of Ravi is 25"

### **File:**

File is a collection of related data stored in secondary memory.

### **File Oriented approach:**

The traditional file oriented approach to information processing has for each application a separate master file and its own set of personal file. In file oriented approach the program dependent on the files and files become dependent on the files and files become dependents upon the programs

### **Disadvantages of file oriented approach:**

1) **Data redundancy and inconsistency:**

The same information may be written in several files. This redundancy leads to higher storage and access cost. It may lead data inconsistency that is the various copies of the same data may longer agree for example a changed customer address may be reflected in single file but not else where in the system.

2) **Difficulty in accessing data :**

The conventional file processing system do not allow data to retrieved in a convenient and efficient manner according to user choice.

3) **Data isolation :**

Because data are scattered in various file and files may be in different formats with new application programs to retrieve the appropriate data is difficult.

4) **Integrity Problems:**

Developers enforce data validation in the system by adding appropriate code in the various application program. How ever when new constraints are added, it is difficult to change the programs to enforce them.

5) **Atomicity:**

It is difficult to ensure atomicity in a file processing system when transaction failure occurs due to power failure, networking problems etc.

(atomicity: either all operations of the transaction are reflected properly in the database or non are)

6) **Concurrent access:**

In the file processing system it is not possible to access a same file for transaction at same time

7) **Security problems:**

There is no security provided in file processing system to secure the data from unauthorized user access.

### **Database:**

A database is organized collection of related data of an organization stored in formatted way which is shared by multiple users.

The main feature of data in a database are:

1. It must be well organized
2. it is related
3. It is accessible in a logical order without any difficulty
4. It is stored only once

for example:

consider the roll no, name, address of a student stored in a student file. It is collection of related data with an implicit meaning.

Data in the database may be persistent, integrated and shared.

**Persistent:**

If data is removed from database due to some explicit request from user to remove.

**Integrated:**

A database can be a collection of data from different files and when any redundancy among those files are removed from database is said to be integrated data.

**Sharing Data:**

The data stored in the database can be shared by multiple users simultaneously with out affecting the correctness of data.

**Why Database:**

In order to overcome the limitation of a file system, a new approach was required. Hence a database approach emerged. A database is a persistent collection of logically related data. The initial attempts were to provide a centralized collection of data. A database has a self describing nature. It contains not only the data sharing and integration of data of an organization in a single database.

A small database can be handled manually but for a large database and having multiple users it is difficult to maintain it, In that case a computerized database is useful. The advantages of database system over traditional, paper based methods of record keeping are:

- **compactness:**  
No need for large amount of paper files
- **speed:**  
The machine can retrieve and modify the data more faster way then human being
- **Less drudgery:** Much of the maintenance of files by hand is eliminated
- **Accuracy:** Accurate, up-to-date information is fetched as per requirement of the user at any time.

**Database Management System (DBMS):**

A database management system consists of collection of related data and refers to a set of programs for defining, creation, maintenance and manipulation of a database.

**Function of DBMS:**

1. **Defining database schema:** it must give facility for defining the database structure also specifies access rights to authorized users.
2. **Manipulation of the database:** The dbms must have functions like insertion of record into database updation of data, deletion of data, retrieval of data
3. **Sharing of database:** The DBMS must share data items for multiple users by maintaining consistency of data.
4. **Protection of database:** It must protect the database against unauthorized users.
5. **Database recovery:** If for any reason the system fails DBMS must facilitate data base recovery.

### **Advantages of dbms:**

#### **Reduction of redundancies:**

Centralized control of data by the DBA avoids unnecessary duplication of data and effectively reduces the total amount of data storage required avoiding duplication in the elimination of the inconsistencies that tend to be present in redundant data files.

#### **Sharing of data:**

A database allows the sharing of data under its control by any number of application programs or users.

#### **Data Integrity:**

Data integrity means that the data contained in the database is both accurate and consistent. Therefore data values being entered for storage could be checked to ensure that they fall within a specified range and are of the correct format.

#### **Data Security:**

The DBA who has the ultimate responsibility for the data in the dbms can ensure that proper access procedures are followed including proper authentication schemas for access to the DBS and additional check before permitting access to sensitive data.

#### **Conflict resolution:**

DBA resolve the conflict on requirements of various user and applications. The DBA chooses the best file structure and access method to get optimal performance for the application.

#### **Data Independence:**

Data independence is usually considered from two points of views; physically data independence and logical data independence.

**Physical data Independence** allows changes in the physical storage devices or organization of the files to be made without requiring changes in the conceptual view or any of the external views and hence in the application programs using the data base.

**Logical data independence** indicates that the conceptual schema can be changed without affecting the existing external schema or any application program.

#### **Disadvantage of DBMS:**

1. DBMS software and hardware (networking installation) cost is high
2. The processing overhead by the dbms for implementation of security, integrity and sharing of the data.
3. centralized database control
4. Setup of the database system requires more knowledge, money, skills, and time.
5. The complexity of the database may result in poor performance.

#### **Database Basics:**

##### **Data item:**

The data item is also called as field in data processing and is the smallest unit of data that has meaning to its users.

Eg: "e101", "sumit"

##### **Entities and attributes:**

An entity is a thing or object in the real world that is distinguishable from all other objects

Eg:

Bank, employee, student

Attributes are properties are properties of an entity.

Eg:

Empcode, ename, rolno, name

##### **Logical data and physical data :**

Logical data are the data for the table created by user in primary memory.

Physical data refers to the data stored in the secondary memory.

##### **Schema and sub-schema :**

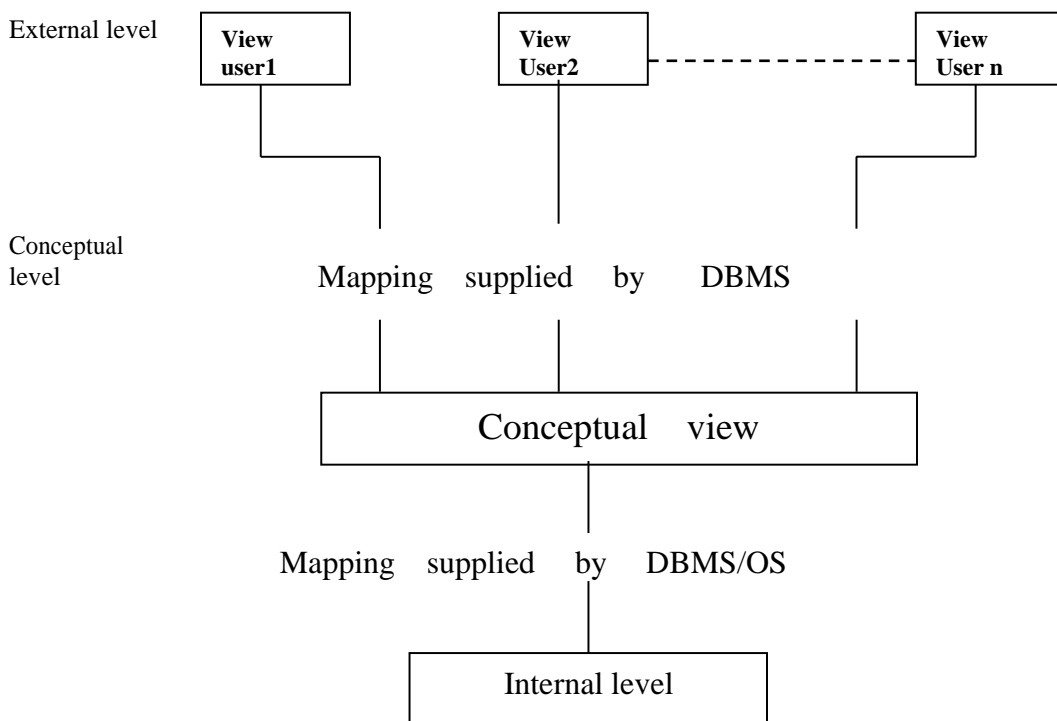
A schema is a logical data base description and is drawn as a chart of the types of data that are used . It gives the names of the entities and attributes and specify the relationships between them.

A database schema includes such information as :

- Characteristics of data items such as entities and attributes .
- Logical structures and relationships among these data items .
- Format for storage representation.
- Integrity parameters such as physical authorization and back up policies.

A *subschema* is derived schema derived from existing schema as per the user requirement. There may be more then one subschema create for a single conceptual schema.

### **Three level architecture of DBMS :**



A database management system that provides three level of data is said to follow three-level architecture .

- External level
- Conceptual level
- Internal level

**External level :**

The external level is at the highest level of database abstraction . At this level, there will be many views define for different users requirement. A view will describe only a subset of the database. Any number of user views may exist for a given global or subschema.

for example , each student has different view of the time table. the view of a student of Btech (CSE) is different from the view of the student of Btech(ECE).Thus this level of abstraction is concerned with different categories of users.

Each external view is described by means of a schema called schema or schema.

### **Conceptual level :**

At this level of database abstraction all the database entities and the relationships among them are included . One conceptual view represents the entire database . This conceptual view is defined by the conceptual schema.

The conceptual schema hides the details of physical storage structures and concentrate on describing entities , data types, relationships, user operations and constraints.

It describes all the records and relationships included in the conceptual view . There is only one conceptual schema per database . It includes feature that specify the checks to relation data consistency and integrity.

### **Internal level :**

It is the lowest level of abstraction closest to the physical storage method used . It indicates how the data will be stored and describes the data structures and access methods to be used by the database . The internal view is expressed by internal schema.

The following aspects are considered at this level:

1. Storage allocation e.g: B-tree,hashing
2. access paths eg. specification of primary and secondary keys,indexes etc
3. Miscellaneous eg. Data compression and encryption techniques,optimization of the internal structures.

### **Database users :**

#### **Naive users :**

Users who need not be aware of the presence of the database system or any other system supporting their usage are considered naïve users . A user of an automatic teller machine falls on this category.



**Online users :**

These are users who may communicate with the database directly via an online terminal or indirectly via a user interface and application program. These users are aware of the database system and also know the data manipulation language system.

**Application programmers :**

Professional programmers who are responsible for developing application programs or user interfaces utilized by the naïve and online user falls into this category.

**Database Administration :**

A person who has central control over the system is called database administrator .  
The function of DBA are :

1. creation and modification of conceptual Schema definition
2. Implementation of storage structure and access method.
3. schema and physical organization modifications .
4. granting of authorization for data access.
5. Integrity constraints specification.
6. Execute immediate recovery procedure in case of failures
7. ensure physical security to database

**Database language :****1) Data definition language(DDL) :**

DDL is used to define database objects .The conceptual schema is specified by a set of definitions expressed by this language. It also give some details about how to implement this schema in the physical devices used to store the data. This definition includes all the entity sets and their associated attributes and their relation ships. The result of DDL statements will be a set of tables that are stored in special file called data dictionary.

**2) Data manipulation language(DML) :**

A DML is a language that enables users to access or manipulate data stored in the database. Data manipulation involves retrieval of data from the database, insertion of new data into the database and deletion of data or modification of existing data.

There are basically two types of DML:

- **procedural:** Which requires a user to specify what data is needed and how to get it.
- **non-rocedural:** which requires a user to specify what data is needed with out specifying how to get it.

### 3) **Data control language(DCL):**

This language enables user to grant authorization and canceling authorization of database objects.

### **Elements of DBMS:**

#### **DML pre-compiler:**

It converts DML statement embedded in an application program to normal procedure calls in the host language. The pre-compiler must interact with the query processor in order to generate the appropriate code.

#### **DDL compiler:**

The DDL compiler converts the data definition statements into a set of tables. These tables contains information concerning the database and are in a form that can be used by other components of the dbms.

#### **File manager:**

File manager manages the allocation of space on disk storage and the data structure used to represent information stored on disk.

#### **Database manager:**

A database manager is a program module which provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.

The responsibilities of database manager are:

1. **Interaction with file manager:** The data is stored on the disk using the file system which is provided by operating system. The database manager translate the the different DML statements into low-level file system commands. so The database manager is responsible for the actual storing, retrieving and updating of data in the database.
2. **Integrity enforcement:** The data values stored in the database must satisfy certain constraints(eg: the age of a person can't be less then zero). These constraints are specified by DBA. Data manager checks the constraints and if it satisfies then it stores the data in the database.
3. **Security enforcement:** Data manager checks the security measures for database from unauthorized users.
4. **Backup and recovery:** Database manager detects the failures occurs due to different causes (like disk failure, power failure, deadlock, s/w error) and restores the database to original state of the database.
5. **Concurrency control:** When several users access the same database file simultaneously, there may be possibilities of data inconsistency. It is

responsible of database manager to control the problems occurs for concurrent transactions.

**query processor:**

The query processor used to interpret to online user’s query and convert it into an efficient series of operations in a form capable of being sent to the data manager for execution. The query processor uses the data dictionary to find the details of data file and using this information it create query plan/access plan to execute the query.

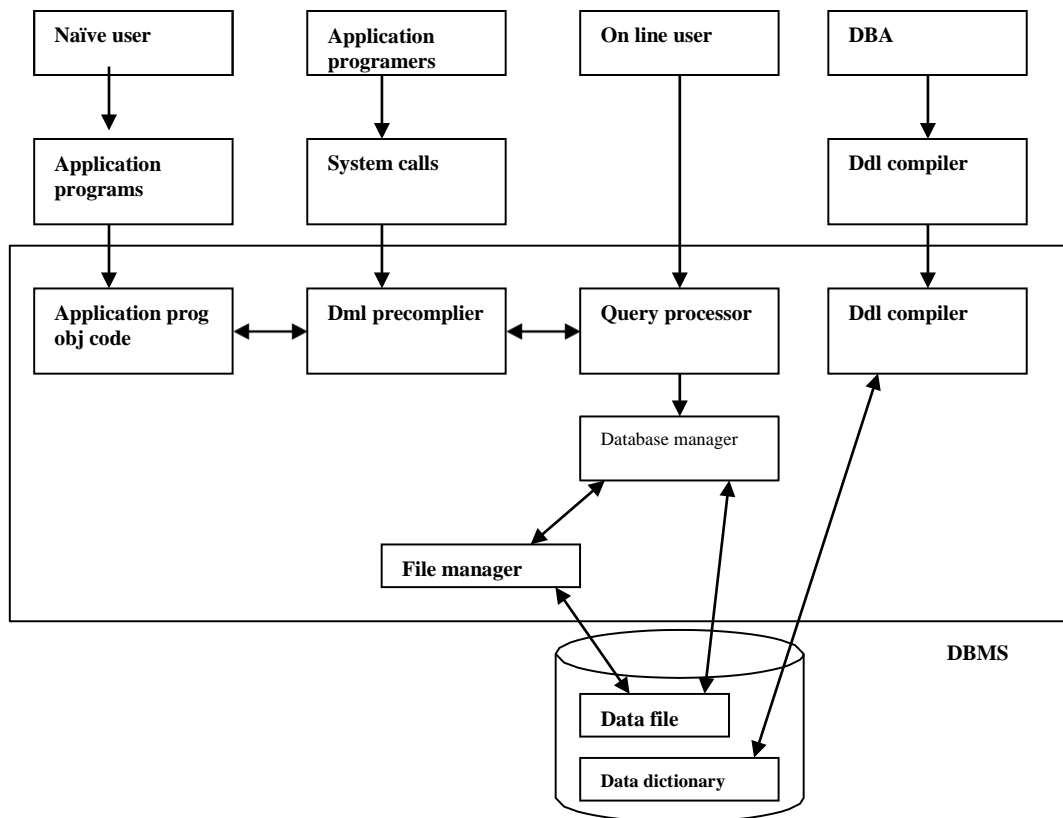
**Data Dictionary:**

Data dictionary is the table which contains the information about database objects. It contains information like

1. external, conceptual and internal database description
2. description of entities , attributes as well as meaning of data elements
3. synonyms, authorization and security codes
4. database authorization

The data stored in the data dictionary is called *meta data*.

**DBMS STRUCTURE:**



**Q. List four significant differences between a file-processing system and a DBMS.**

Answer: Some main differences between a database management system and a file-processing system are:

- Both systems contain a collection of data and a set of programs which access that data. A database management system coordinates both the physical and the logical

access to the data, whereas a file-processing system coordinates only the physical access.

- A database management system reduces the amount of data duplication by ensuring that a physical piece of data is available to all programs authorized to have access to it, whereas data written by one program in a file-processing system may not be readable by another program.
- A database management system is designed to allow flexible access to data (i.e., queries), whereas a file-processing system is designed to allow predetermined access to data (i.e., compiled programs).
- A database management system is designed to coordinate multiple users accessing the same data at the same time. A file-processing system is usually designed to allow one or more programs to access different data files at the same time. In a file-processing system, a file can be accessed by two programs concurrently only if both programs have read-only access to the file.

**Q.Explain the difference between physical and logical data independence.**

Answer:

- Physical data independence is the ability to modify the physical scheme without making it necessary to rewrite application programs. Such modifications include changing from unblocked to blocked record storage, or from sequential to random access files.
- Logical data independence is the ability to modify the conceptual scheme without making it necessary to rewrite application programs. Such a modification might be adding a field to a record; an application program's view hides this change from the program.

**Q. List five responsibilities of a database management system. For each responsibility, explain the problems that would arise if the responsibility were not discharged.**

Answer: A general purpose database manager (DBM) has five responsibilities:

- a. interaction with the file manager.
- b. integrity enforcement.
- c. security enforcement.
- d. backup and recovery.
- e. concurrency control.

If these responsibilities were not met by a given DBM (and the text points out that sometimes a responsibility is omitted by design, such as concurrency control on a single-user DBM for a micro computer) the following problems can occur, respectively:

- a. No DBM can do without this, if there is no file manager interaction then nothing stored in the files can be retrieved.

- b. Consistency constraints may not be satisfied, account balances could go below the minimum allowed, employees could earn too much overtime (e.g., hours > 80) or, airline pilots may fly more hours than allowed by law.
- c. Unauthorized users may access the database, or users authorized to access part of the database may be able to access parts of the database for which they lack authority. For example, a high school student could get access to national defense secret codes, or employees could find out what their supervisors earn.
- d. Data could be lost permanently, rather than at least being available in a consistent state that existed prior to a failure.
- e. Consistency constraints may be violated despite proper integrity enforcement in each transaction. For example, incorrect bank balances might be reflected due to simultaneous withdrawals and deposits, and so on.

**Q. What are five main functions of a database administrator?**

Answer: Five main functions of a database administrator are:

- To create the scheme definition
- To define the storage structure and access methods
- To modify the scheme and/or physical organization when necessary
- To grant authorization for data access
- To specify integrity constraints

**Q. List six major steps that you would take in setting up a database for a particular enterprise.**

Answer: Six major steps in setting up a database for a particular enterprise are:

- Define the high level requirements of the enterprise (this step generates a document known as the system requirements specification.)
- Define a model containing all appropriate types of data and data relationships.
- Define the integrity constraints on the data.
- Define the physical level.
- For each known problem to be solved on a regular basis (e.g., tasks to be carried out by clerks or Web users) define a user interface to carry out the task, and write the necessary application programs to implement the user interface.
- Create/initialize the database.

**EXERCISES:**

1. What is database management system
2. What are the disadvantage of file processing system

3. State advantage and disadvantage of database management system
4. What are different types of database users
5. What is data dictionary and what are its contents
6. What are the functions of DBA
7. What are the different database languages explain with example.
8. Explain the three layer architecture of DBMS.
9. Differentiate between physical data independence and logical data independence
10. Explain the function of database manager
11. Explain meta data

## **CHAPTER-2**

### **ER-MODEL**

#### **Data model:**

The data model describes the structure of a database. It is a collection of conceptual tools for describing data, data relationships and consistency constraints and various types of data model such as

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

Types of data model:

1. Object based logical model
  - a. ER-model
  - b. Functional model
  - c. Object oriented model
  - d. Semantic model
2. Record based logical model
  - a. Hierarchical database model
  - b. Network model
  - c. Relational model
3. Physical model

### **Entity Relationship Model**

The entity-relationship data model perceives the real world as consisting of basic objects, called entities and relationships among these objects. It was developed to facilitate data base design by allowing specification of an enterprise schema which represents the overall logical structure of a data base.

#### **Main features of ER-MODEL:**

- Entity relationship model is a high level conceptual model
- It allows us to describe the data involved in a real world enterprise in terms of objects and their relationships.
- It is widely used to develop an initial design of a database
- It provides a set of useful concepts that make it convenient for a developer to move from a baseid set of information to a detailed and description of information that can be easily implemented in a database system
- It describes data as a collection of entities, relationships and attributes.

## **Basic concepts:**

The E-R data model employs three basic notions : entity sets, relationship sets and attributes.

### **Entity sets:**

An entity is a “thing” or “object” in the real world that is distinguishable from all other objects. For example, each person in an enterprise is an entity. An entity has a set properties and the values for some set of properties may uniquely identify an entity.

BOOK is entity and its properties(called as attributes) bookcode, booktitle, price etc .

An entity set is a set of entities of the same type that share the same properties, or attributes. The set of all persons who are customers at a given bank, for example, can be defined as the entity set customer.

### **Attributes:**

An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set.

**Customer** is an entity and its attributes are **customerid, custmername, custaddress** etc.

An attribute as used in the E-R model , can be characterized by the following attribute types.

#### **a) Simple and composite attribute:**

simple attributes are the attributes which can't be divided into sub parts

eg: customerid, empno

composite attributes are the attributes which can be divided into subparts.

eg: name consisting of first name, middle name, last name

address consisting of city, pincode, state

#### **b) single-valued and multi-valued attribute:**

The attribute having unique value is single –valued attribute

eg: empno, customerid, regdno etc.

The attribute having more than one value is multi-valued attribute

eg: phone-no, dependent name, vehicle

#### **c) Derived Attribute:**

The values for this type of attribute can be derived from the values of existing attributes

eg: age which can be derived from (currentdate-birthdate)

experience\_in\_year can be calculated as (currentdate-joindate)

#### **d) NULL valued attribute:**

The attribute value which is unknown to user is called NULL valued attribute.



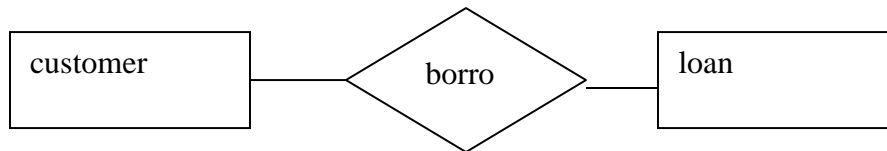
### Relationship sets:

A relationship is an association among several entities.

A relationship set is a set of relationships of the same type. Formally, it is a mathematical relation on  $n \geq 2$  entity sets. If  $E_1, E_2, \dots, E_n$  are entity sets, then a relationship set  $R$  is a subset of

$\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$

where  $(e_1, e_2, \dots, e_n)$  is a relationship.



Consider the two entity sets customer and loan. We define the relationship set borrow to denote the association between customers and the bank loans that the customers have.

### Mapping Cardinalities:

Mapping cardinalities or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.

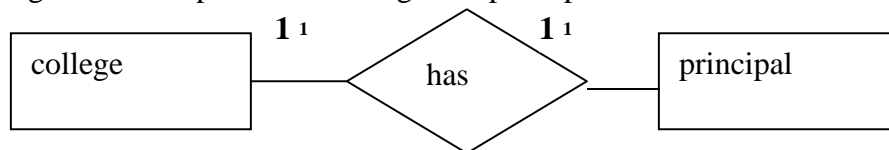
Mapping cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets.

For a binary relationship set  $R$  between entity sets  $A$  and  $B$ , the mapping cardinalities must be one of the following:

#### **one to one:**

An entity in  $A$  is associated with at most one entity in  $B$ , and an entity in  $B$  is associated with at most one entity in  $A$ .

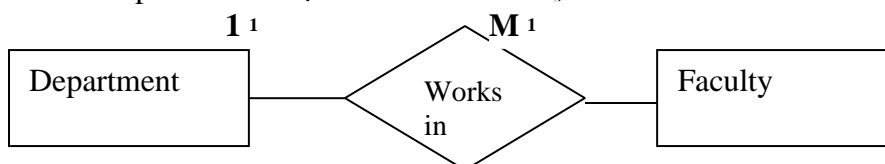
Eg: relationship between college and principal



#### **One to many:**

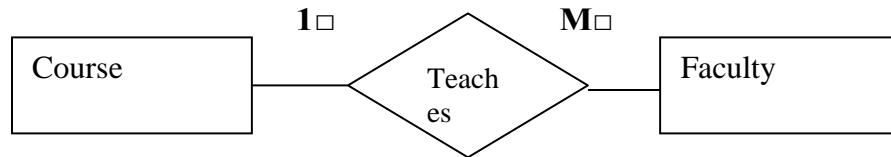
An entity in  $A$  is associated with any number of entities in  $B$ . An entity in  $B$  is associated with at the most one entity in  $A$ .

Eg: Relationship between department and faculty

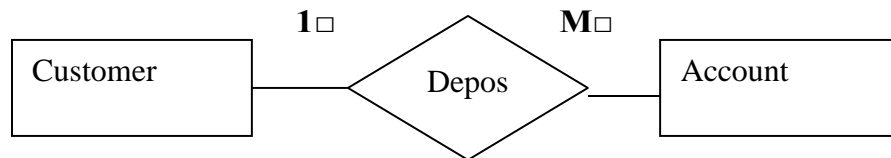


**Many to one:**

An entity in A is associated with at most one entity in B. An entity in B is associated with any number in A.

**Many –to-many:**

Entities in A and B are associated with any number of entities from each other.

**More about entities and Relationship:****Recursive relationships:**

When the same entity type participates more than once in a relationship type in different roles, the relationship types are called recursive relationships.

**Participation constraints:**

The participation constraints specify whether the existence of any entity depends on its being related to another entity via the relationship. There are two types of participation constraints

**Total :**

.When all the entities from an entity set participate in a relationship type , is called total participation. For example, the participation of the entity set student on the relationship set must 'opts' is said to be total because every student enrolled must opt for a course.

**Partial:**

When it is not necessary for all the entities from an entity set to participate in a relationship type, it is called participation. For example, the participation of the entity set student in 'represents' is partial, since not every student in a class is a class representative.

**Weak Entity:**

Entity types that do not contain any key attribute, and hence can not be identified independently are called weak entity types. A weak entity can be identified by uniquely only by considering some of its attributes in conjunction with the primary key attribute of another entity, which is called the identifying owner entity.

Generally a partial key is attached to a weak entity type that is used for unique identification of weak entities related to a particular owner type. The following restrictions must hold:

- The owner entity set and the weak entity set must participate in one to many relationship set. This relationship set is called the identifying relationship set of the weak entity set.

- The weak entity set must have total participation in the identifying relationship.

**Example:**

Consider the entity type dependent related to employee entity, which is used to keep track of the dependents of each employee. The attributes of dependents are : name ,birthrate, sex and relationship. Each employee entity set is said to its own the dependent entities that are related to it. However, not that the 'dependent' entity does not exist of its own., it is dependent on the employee entity. In other words we can say that in case an employee leaves the organization all dependents related to without the entity 'employee'. Thus it is a weak entity.

**Keys:**

**Super key:**

A super key is a set of one or more attributes that taken collectively, allow us to identify uniquely an entity in the entity set.

For example , customer-id,(cname,customer-id),(cname,telno)

**Candidate key:**

In a relation R, a candidate key for R is a subset of the set of attributes of R, which have the following properties:

- *Uniqueness:* no two distinct tuples in R have the same values for the candidate key
- *Irreducible:* No proper subset of the candidate key has the uniqueness property that is the candidate key.

Eg: (cname,telno)

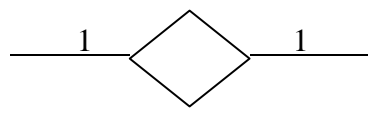
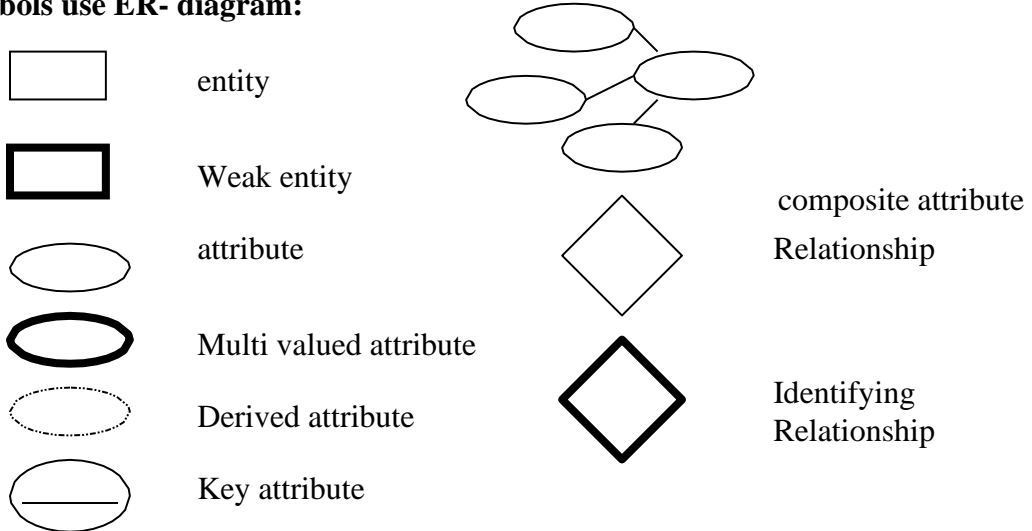
**Primary key:**

The primary key is the candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. The remaining candidate keys if any, are called *alternate key*.

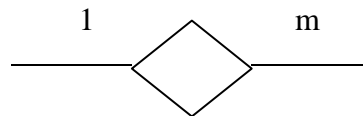
**ER-DIAGRAM:**

The overall logical structure of a database using ER-model graphically with the help of an ER-diagram.

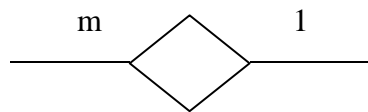
**Symbols use ER- diagram:**



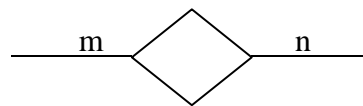
One-to -one



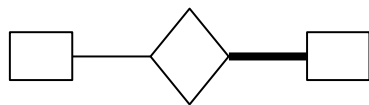
One-to -many



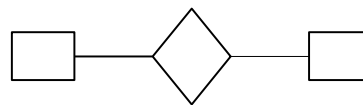
many-to -one



many-to -many



Total participation



Partial participation

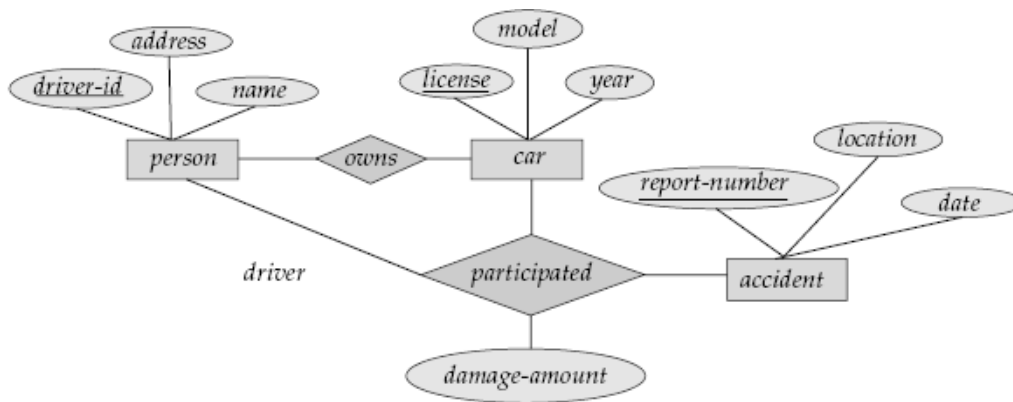


Figure 2.1 E-R diagram for a Car-insurance company.

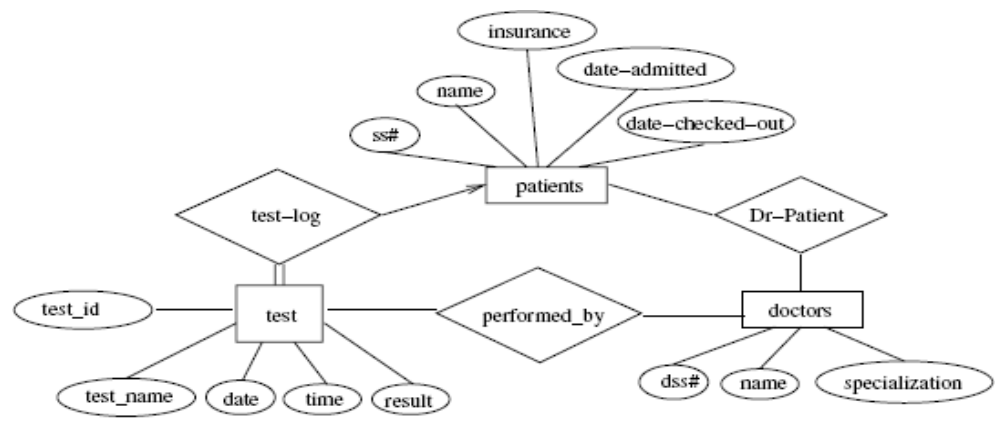


Figure 2.2 E-R diagram for a hospital.

A university registrar's office maintains data about the following entities: (a) courses, including number, title, credits, syllabus, and prerequisites; (b) course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom; (c) students, including student-id, name, and program; and (d) instructors, including identification number, name, department, and title. Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled.

Construct an E-R diagram for the registrar's office. Document all assumptions that you make about the mapping constraints.

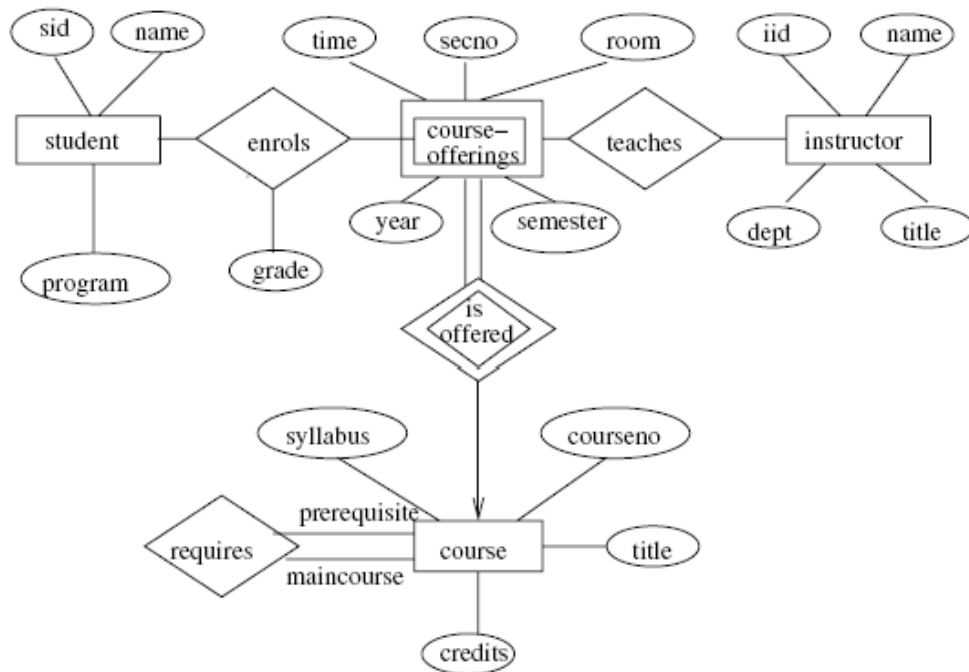


Figure 2.3 E-R diagram for a university.

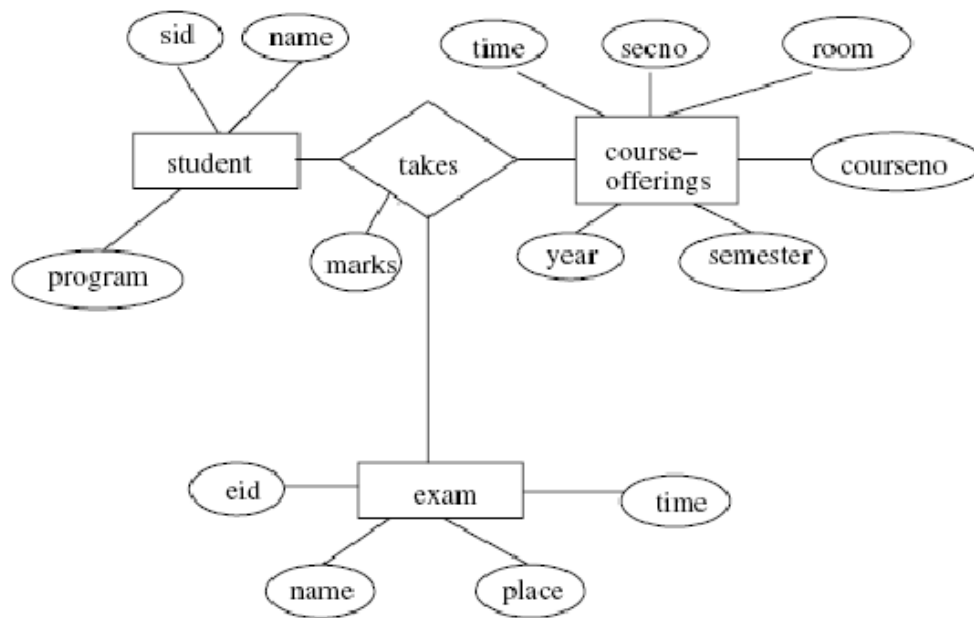


Figure 2.4 E-R diagram for marks database.

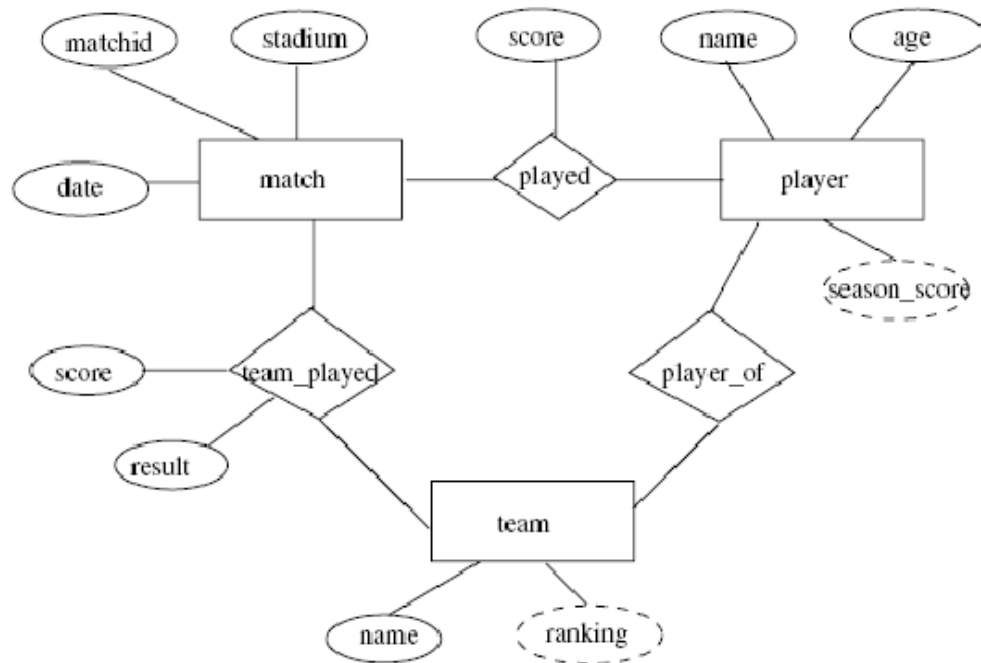


Figure 2.7 E-R diagram for all teams statistics.

Consider a university database for the scheduling of classrooms for final exams. This database could be modeled as the single entity set *exam*, with attributes *course-name*, *section-number*, *room-number*, and *time*. Alternatively, one or more additional entity sets could be defined, along with relationship sets to replace some of the attributes of the *exam* entity set, as

- *course* with attributes *name*, *department*, and *c-number*
- *section* with attributes *s-number* and *enrollment*, and dependent as a weak entity set on *course*
- *room* with attributes *r-number*, *capacity*, and *building*

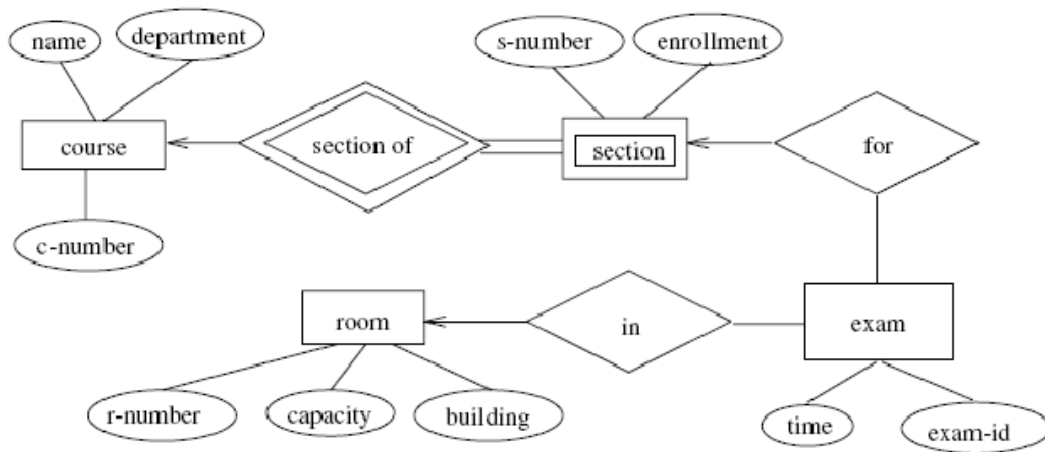
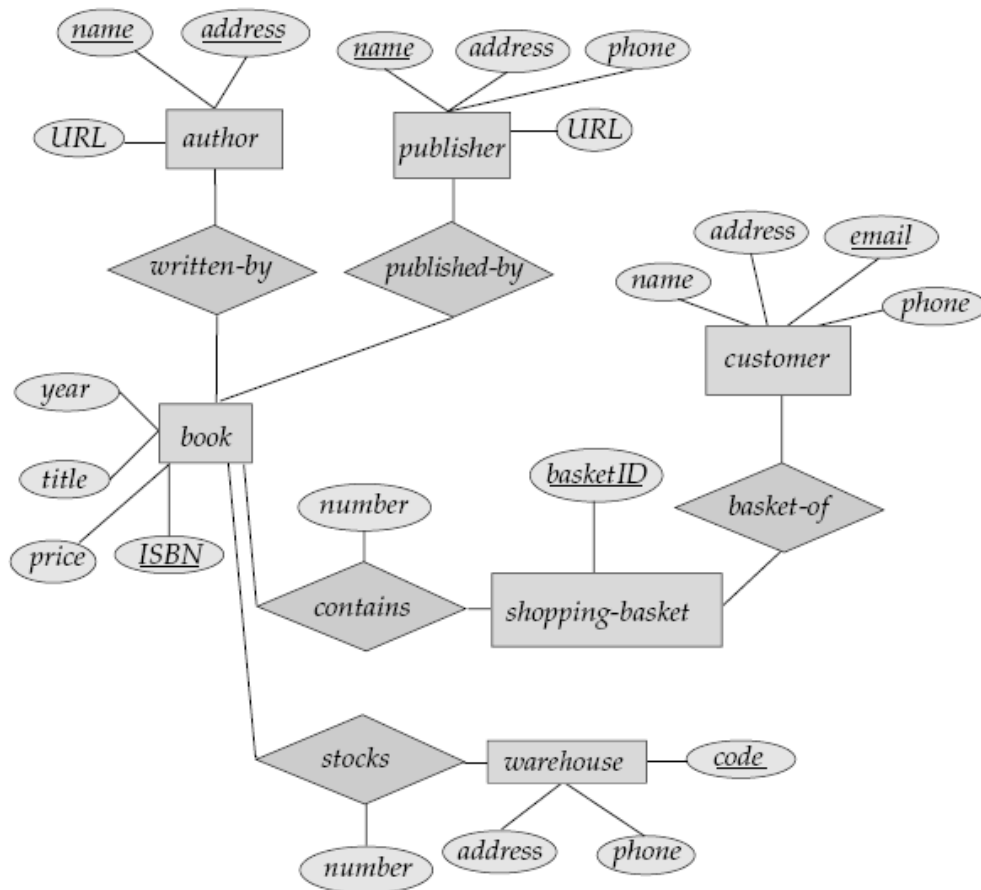


Figure 2.12 E-R diagram for exam scheduling.





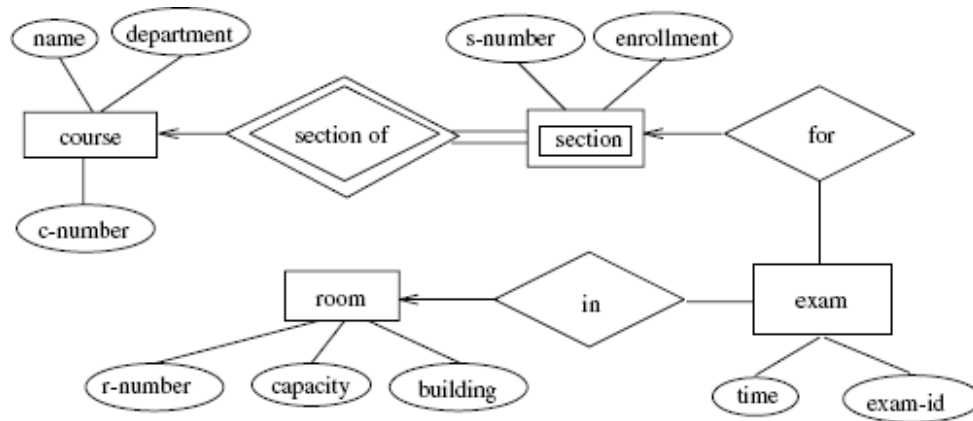


Figure 2.12 E-R diagram for exam scheduling.

### Advanced ER-diagram:

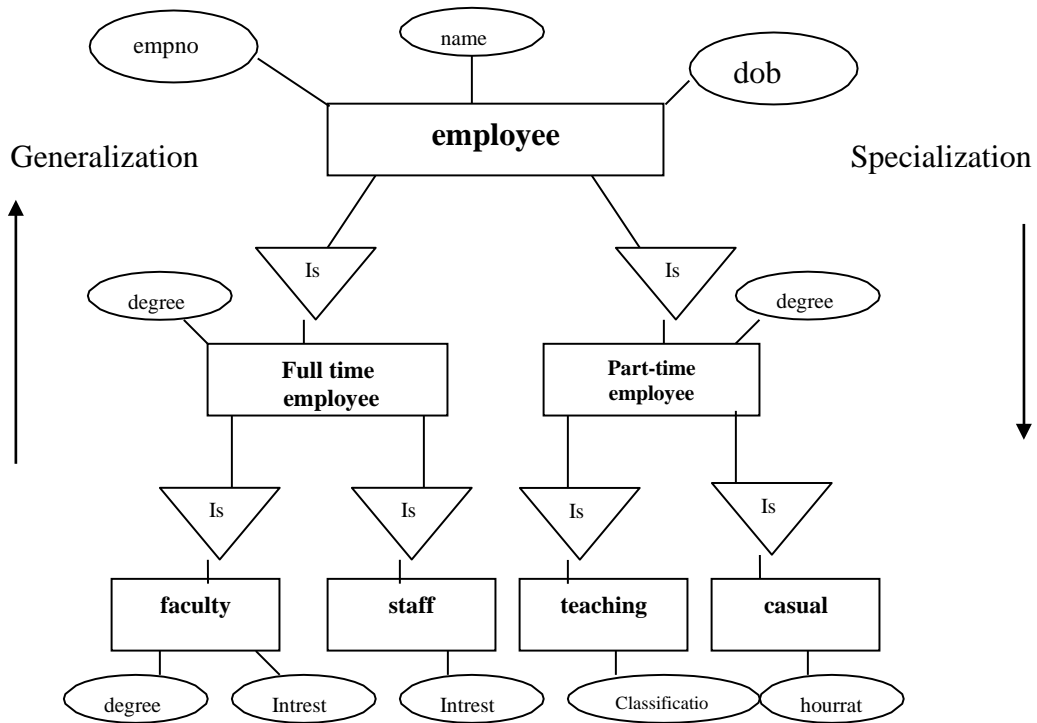
Abstraction is the simplification mechanism used to hide superfluous details of a set of objects. It allows one to concentrate on the properties that are of interest to the application.

There are two main abstraction mechanism used to model information:

#### Generalization and specialization:

*Generalization* is the abstracting process of viewing set of objects as a single general class by concentrating on the general characteristics of the constituent sets while suppressing or ignoring their differences. It is the union of a number of lower-level entity types for the purpose of producing a higher-level entity type. For instance, student is a generalization of graduate or undergraduate, full-time or part-time students. Similarly, employee is generalization of the classes of objects cook, waiter, and cashier. Generalization is an IS\_A relationship; therefore, manager IS\_AN employee, cook IS\_AN employee, waiter IS\_AN employee, and so forth.

*Specialization* is the abstracting process of introducing new characteristics to an existing class of objects to create one or more new classes of objects. This involves taking a higher-level, and using additional characteristics, generating lower-level entities. The lower-level entities also inherits the, characteristics of the higher-level entity. In applying the characteristics size to car we can create a full-size, mid-size, compact or subcompact car. Specialization may be seen as the reverse process of generalization addition specific properties are introduced at a lower level in a hierarchy of objects.



EMPLOYEE(**empno**,name,dob)  
 FULL\_TIME\_EMPLOYEE(**empno**,sala  
 ry)  
 PART\_TIME\_EMPLOYEE(**empno**,type)

Faculty(**empno**,degree,intrest)  
 Staff(**empno**,hour-rate)  
 Teaching (**empno**,stipend)

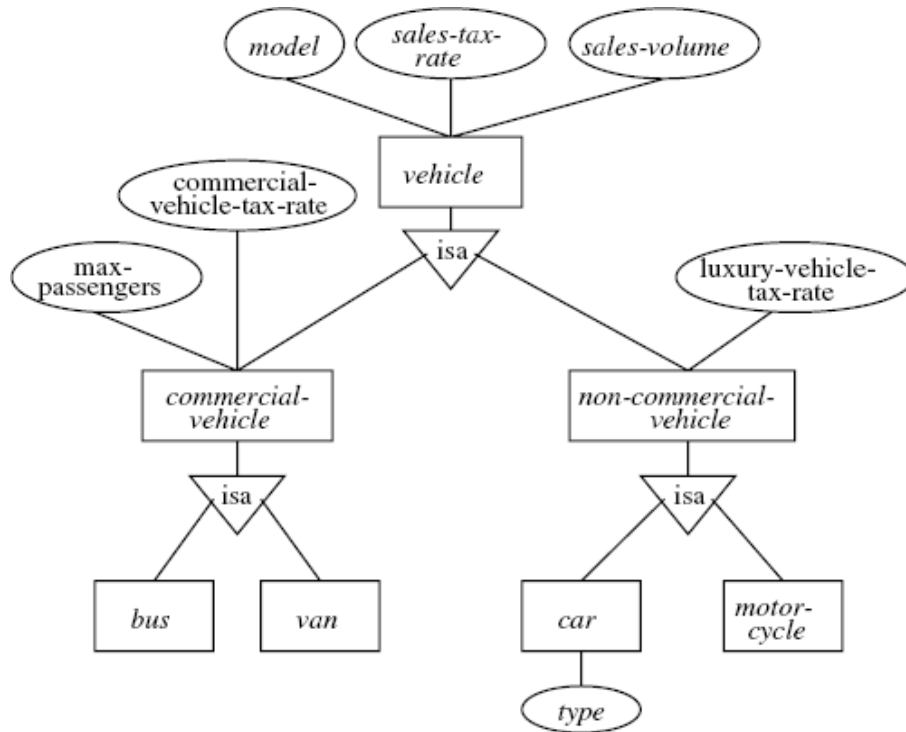
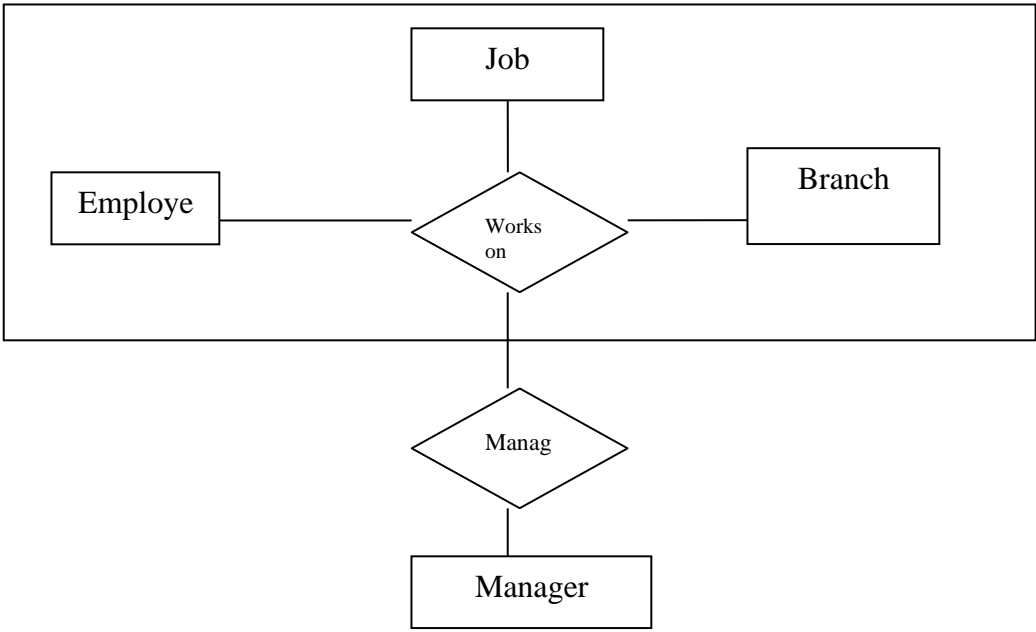


Figure 2.19 E-R diagram of motor-vehicle sales company.

**Aggregation:**

Aggregation is the process of compiling information on an object, there by abstracting a higher level object. In this manner, the entity person is derived by aggregating the characteristics of name, address, ssn. Another form of the aggregation is abstracting a relationship objects and viewing the relationship as an object.



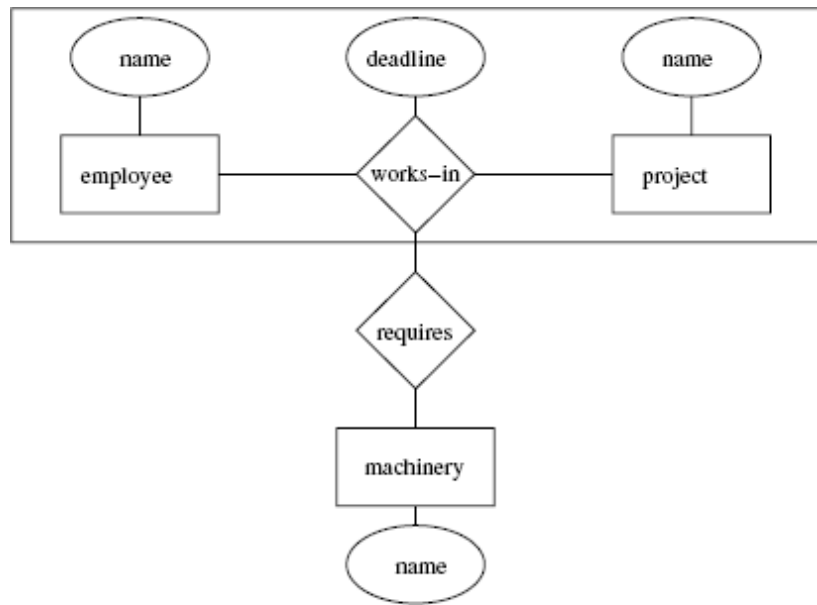


Figure 2.8 E-R diagram Example 1 of aggregation.

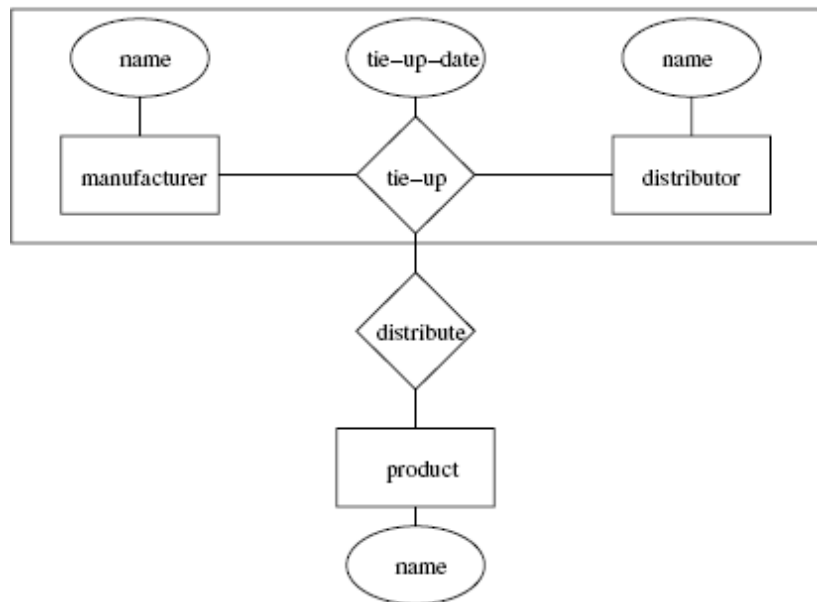
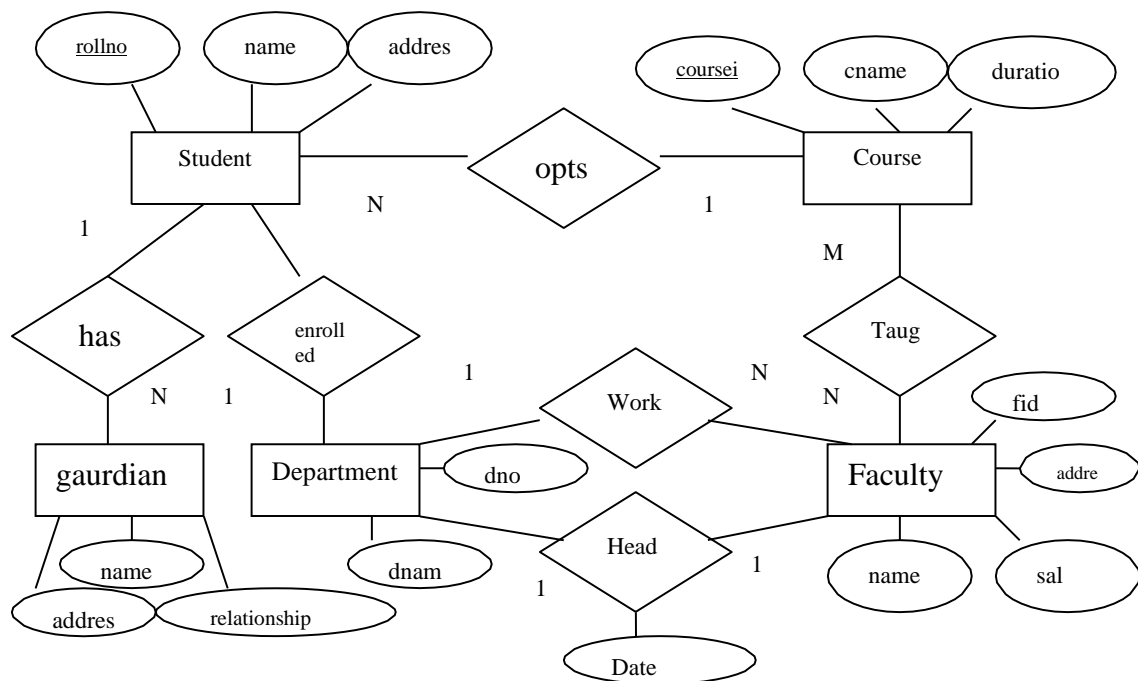


Figure 2.9 E-R diagram Example 2 of aggregation.

Explain the distinctions among the terms primary key, candidate key, and superkey.

**Answer:** A *superkey* is a set of one or more attributes that, taken collectively, allows us to identify uniquely an entity in the entity set. A superkey may contain extraneous attributes. If  $K$  is a superkey, then so is any superset of  $K$ . A superkey for which no proper subset is also a superkey is called a *candidate key*. It is possible that several distinct sets of attributes could serve as candidate keys. The *primary key* is one of the candidate keys that is chosen by the database designer as the principal means of identifying entities within an entity set.

### ER- Diagram For College Database



### Conversion of ER-diagram to relational database

#### Conversion of entity sets:

1. For each strong entity type  $E$  in the ER diagram, we create a relation  $R$  containing all the single attributes of  $E$ . The primary key of the relation  $R$  will be one of the key attribute of  $R$ .

**STUDENT**(rollno (primary key),name, address)

**FACULTY**(id(primary key),name ,address, salary)

**COURSE**(course-id,(primary key),course\_name,duration)

**DEPARTMENT**(dno(primary key),dname)

- for each weak entity type W in the ER diagram, we create another relation R that contains all simple attributes of W. If E is an owner entity of W then key attribute of E is also include In R. This key attribute of R is set as a foreign key attribute of R. Now the combination of primary key attribute of owner entity type and partial key of the weak entity type will form the key of the weak entity type

GUARDIAN((rollno,name) (primary key),address,relationship)

### **Conversion of relationship sets:**

#### **Binary Relationships:**

- **One-to-one relationship:**

For each 1:1 relationship type R in the ER-diagram involving two entities E1 and E2 we choose one of entities(say E1) preferably with total participation and add primary key attribute of another E as a foreign key attribute in the table of entity(E1). We will also include all the simple attributes of relationship type R in E1 if any, For example, the department relationship has been extended tp include head-id and attribute of the relationship.

DEPARTMENT(D\_NO,D\_NAME,HEAD\_ID,DATE\_FROM)

- **One-to-many relationship:**

For each 1:n relationship type R involving two entities E1 and E2, we identify the entity type (say E1) at the n-side of the relationship type R and include primary key of the entity on the other side of the relation (say E2) as a foreign key attribute in the table of E1. We include all simple attribute(or simple components of a composite attribute of R(if any) in he table E1)

For example:

The works in relationship between the DEPARTMENT and FACULTY. For this relationship choose the entity at N side, i.e, FACULTY and add primary key attribute of another entity DEPARTMENT, ie, DNO as a foreign key attribute in FACULTY.

FACULTY(CONSTAINS WORKS\_IN RELATIOSHIP)  
(ID,NAME,ADDRESS,BASIC\_SAL,DNO)

- **Many-to-many relationship:**

For each m:n relationship type R, we create a new table (say S) to represent R, We also include the primary key attributes of both the participating entity types as a foreign key attribute in s. Any simple attributes of the m:n relationship type(or simple components as a composite attribute) is also included as attributes of S.

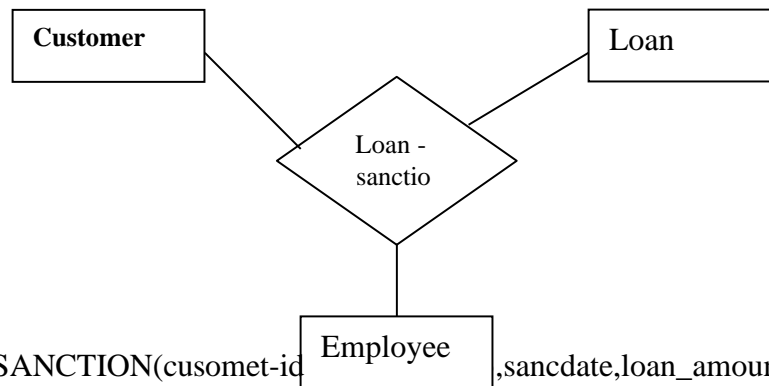
For example:

The M:n relationship taught-by between entities COURSE; and FACULTY shod be represented as a new table. The structure of the table will include primary key of COURSE and primary key of FACULTY entities.

TAUGHT-BY(ID (primary key of FACULTY table),course-id (primary key of COURSE table))

- **N-ary relationship:**

For each n-ary relationship type R where  $n > 2$ , we create a new table S to represent R. We include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. We also include any simple attributes of the n-ary relationship type (or simple components of complete attribute) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types.



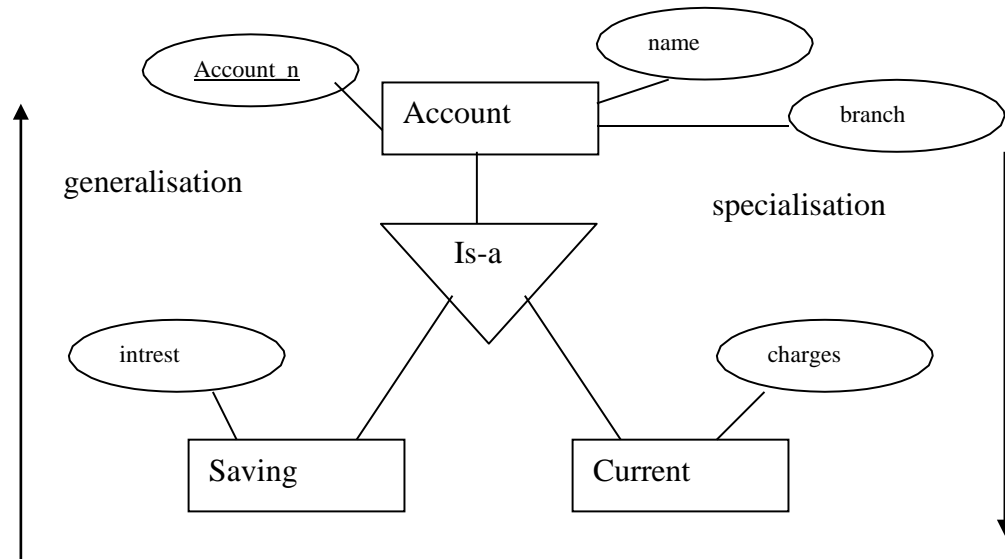
- **Multi-valued attributes:**

For each multivalued attribute 'A', we create a new relation R that includes an attribute corresponding to plus the primary key attributes k of the relation that represents the entity type or relationship that has as an attribute. The primary key of R is then combination of A and k.

For example, if a STUDENT entity has rollno,name and phone number where phone number is a multivalued attribute the we will create table PHONE(rollno,phoneno) where primary key is the combination,In the STUDENT table we need not have phone number, instead if can be simply (rollno,name) only.

PHONE(rollno,phoneno)





- **Converting Generalisation /specification hierarchy to tables:**

A simple rule for conversion may be to decompose all the specialized entities into table in case they are disjoint, for example, for the figure we can create the two table as:

Account(account\_no,name,branch,balance)

Saving account(account-no,intrest)

Current\_account(account-no,charges)

## Record Based Logical Model

### Hierarchical Model:

- A hierarchical database consists of a collection of *records* which are connected to one another through *links*.
- a record is a collection of fields, each of which contains only one data value.
- A link is an association between precisely two records.
- The hierarchical model differs from the network model in that the records are organized as collections of trees rather than as arbitrary graphs.

### **Tree-Structure Diagrams:**

- The schema for a hierarchical database consists of
  - *boxes*, which correspond to record types
  - *lines*, which correspond to links
- Record types are organized in the form of a *rooted tree*.
  - No cycles in the underlying graph.
  - Relationships formed in the graph must be such that only one-to-many or one-to-one relationships exist between a parent and a child.

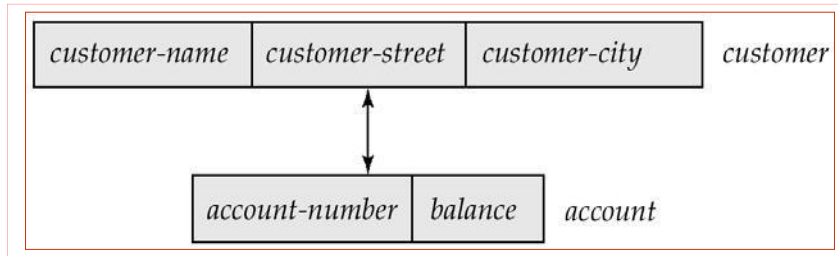
Database schema is represented as a collection of tree-structure diagrams.

- *single* instance of a database tree
- The root of this tree is a dummy node
- The children of that node are actual instances of the appropriate record type

When transforming E-R diagrams to corresponding tree-structure diagrams, we must ensure that the resulting diagrams are in the form of rooted trees.

### **Single Relationships:**

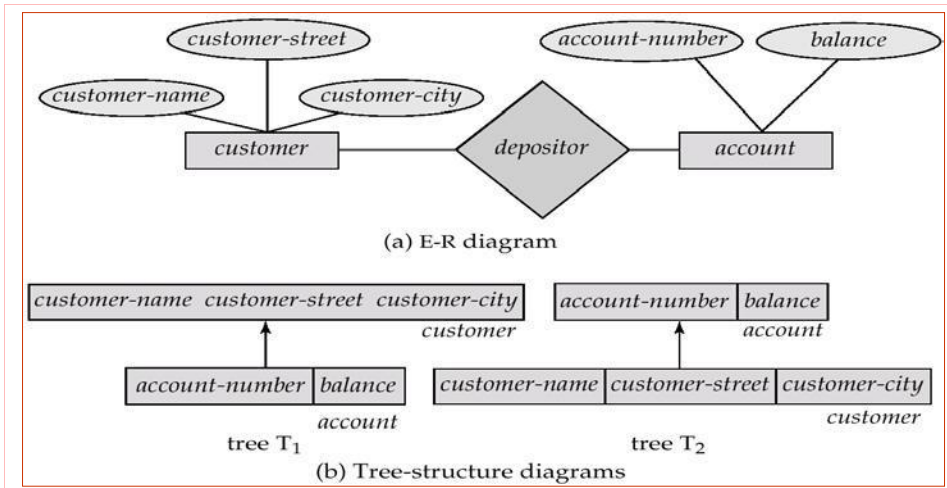
- Example E-R diagram with two entity sets, *customer* and *account*, related through a binary, one-to-many relationship *depositor*.
- Corresponding tree-structure diagram has
  - the record type *customer* with three fields: *customer-name*, *customer-street*, and *customer-city*.
  - the record type *account* with two fields: *account-number* and *balance*
  - the link *depositor*, with an arrow pointing to *customer*
- If the relationship *depositor* is one to one, then the link *depositor* has two arrows.



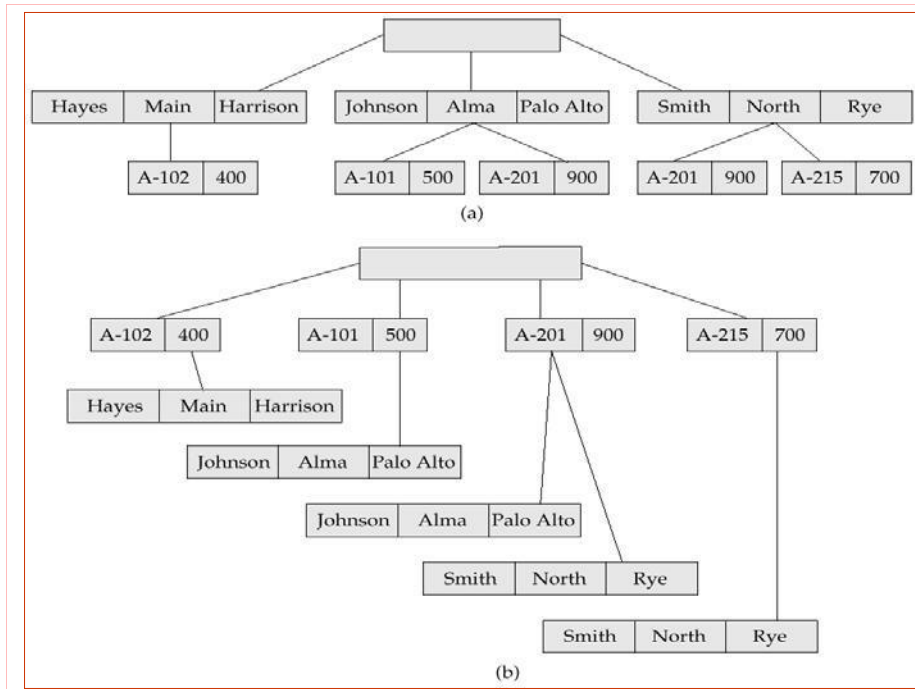
- Only one-to-many and one-to-one relationships can be directly represented in the hierarchical mode.

### Transforming Many-To-Many Relationships:

- Must consider the type of queries expected and the degree to which the database schema fits the given E-R diagram.
- In all versions of this transformation, the underlying database tree (or trees) will have replicated records.



- Create two tree-structure diagrams,  $T_1$ , with the root *customer*, and  $T_2$ , with the root *account*.
- In  $T_1$ , create *depositor*, a many-to-one link from *account* to *customer*.
- In  $T_2$ , create *account-customer*, a many-to-one link from *customer* to *account*.



### Virtual Records:

- For many-to-many relationships, record replication is necessary to preserve the tree-structure organization of the database.
  - Data inconsistency may result when updating takes place
  - Waste of space is unavoidable
- *Virtual record* — contains no data value, only a logical pointer to a particular physical record.
- When a record is to be replicated in several database trees, a single copy of that record is kept in one of the trees and all other records are replaced with a virtual record.
- Let  $R$  be a record type that is replicated in  $T_1, T_2, \dots, T_n$ . Create a new virtual record type *virtual-R* and replace  $R$  in each of the  $n - 1$  trees with a record of type *virtual-R*.
- Eliminate data replication in the diagram shown on page B.11; create *virtual-customer* and *virtual-account*.
- Replace *account* with *virtual-account* in the first tree, and replace *customer* with *virtual-customer* in the second tree.
- Add a dashed line from *virtual-customer* to *customer*, and from *virtual-account* to *account*, to specify the association between a virtual record and its corresponding physical record.

### Network Model:

- Data are represented by collections of *records*.
  - similar to an entity in the E-R model
  - Records and their fields are represented as *record type*
- type *customer* = record                      type *account* = record type
 

<i>customer-name</i> : string;	<i>account-number</i> : integer;
<i>customer-street</i> : string;	<i>balance</i> : integer;

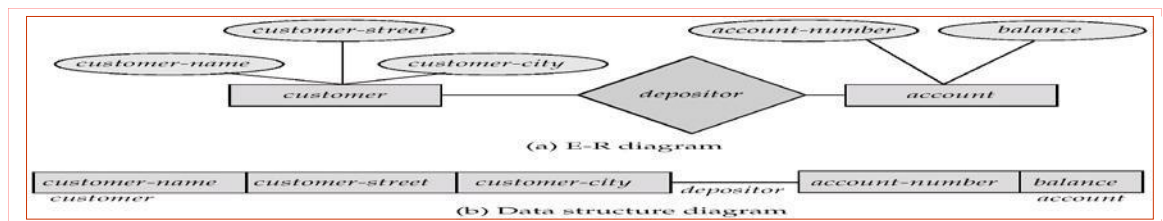
*customer-city*: string;

- end
- Relationships among data are represented by *links*
  - similar to a restricted (binary) form of an E-R relationship
  - restrictions on links depend on whether the relationship is many-many, many-to-one, or one-to-one.

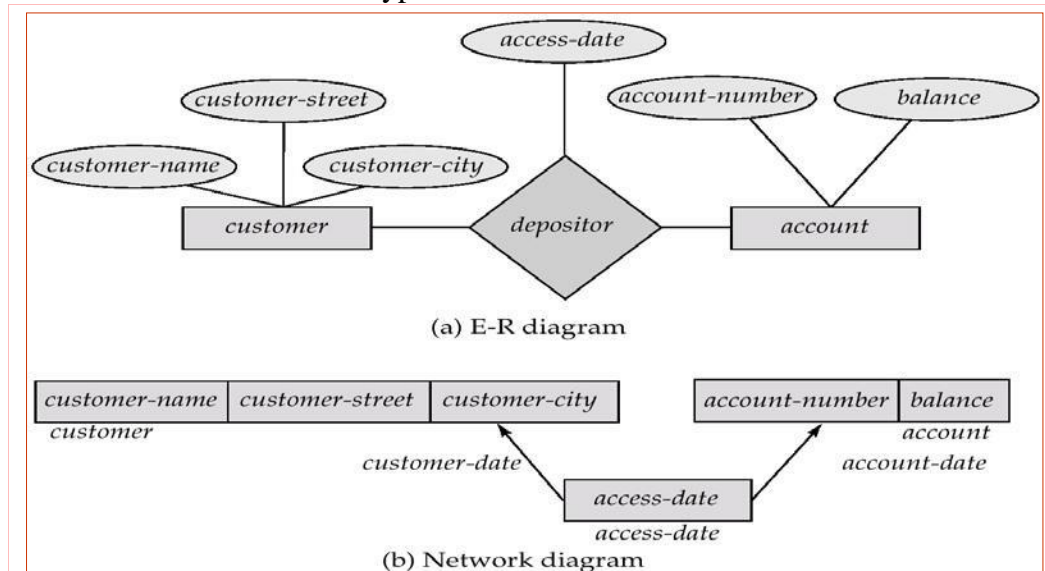
### Data-Structure Diagrams:

- Schema representing the design of a network database.
- A data-structure diagram consists of two basic components:
  - Boxes, which correspond to record types.
  - Lines, which correspond to links.
- Specifies the overall logical structure of the database.

For every E-R diagram, there is a corresponding data-structure diagram.

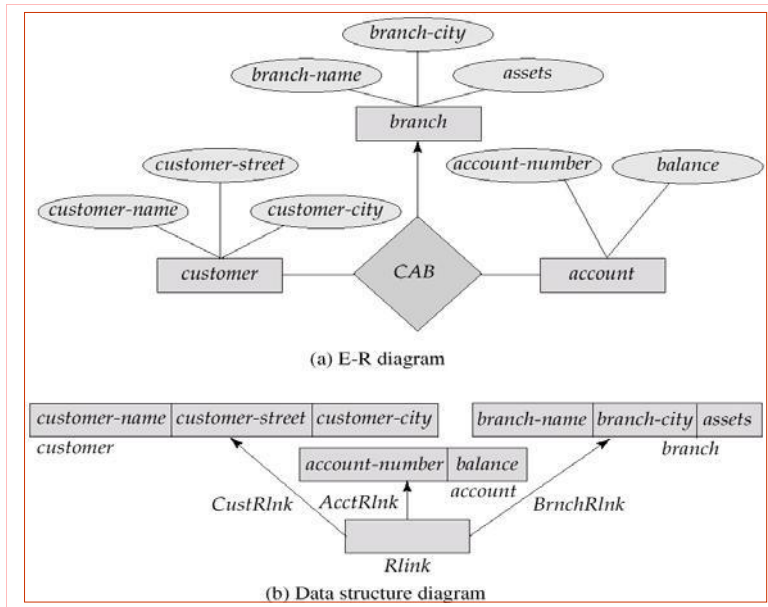


Since a link cannot contain any data value, represent an E-R relationship with attributes with a new record type and links.



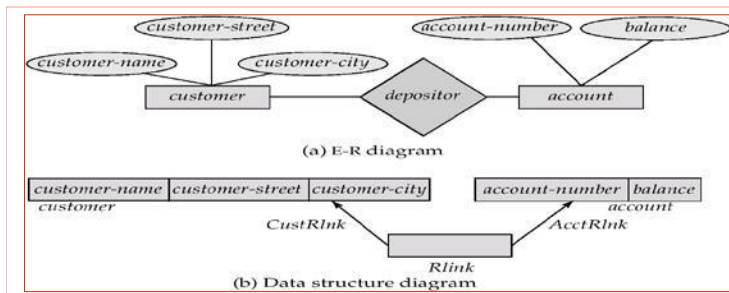
To represent an E-R relationship of degree 3 or higher, connect the participating record types through a new record type that is linked directly to each of the original record types.

1. Replace entity sets *account*, *customer*, and *branch* with record types *account*, *customer*, and *branch*, respectively.
2. Create a new record type *Rlink* (referred to as a *dummy* record type).
3. Create the following many-to-one links:
  - *CustRlink* from *Rlink* record type to *customer* record type
  - *AcctRlnk* from *Rlink* record type to *account* record type
  - *BrncRlnk* from *Rlink* record type to *branch* record type



### The DBTG CODASYL Model:

- All links are treated as many-to-one relationships.
- To model many-to-many relationships, a record type is defined to represent the relationship and two links are used.



### DBTG Sets:

- The structure consisting of two record types that are linked together is referred to in the DBTG model as a *DBTG set*
- In each DBTG set, one record type is designated as the *owner*, and the other is designated as the *member*, of the set.
- Each DBTG set can have any number of *set occurrences* (actual instances of linked records).
- Since many-to-many links are disallowed, each set occurrence has precisely one owner, and has zero or more member records.
- No member record of a set can participate in more than one occurrence of the set at any point.
- A member record can participate simultaneously in several set occurrences of *different* DBTG sets.

## RELATIONAL MODEL

Relational model is simple model in which database is represented as a collection of “relations” where each relation is represented by two-dimensional table.

<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

The relational model was founded by E.F.Codd of the IBM in 1972. The basic concept in the relational model is that of a relation.

### **Properties:**

- It is column homogeneous. In other words, in any given column of a table, all items are of the same kind.
- Each item is a simple number or a character string. That is a table must be in first normal form.
- All rows of a table are distinct.
- The ordering of rows within a table is immaterial.
- The columns of a table are assigned distinct names and the ordering of these columns is immaterial.

### **Domain, attributes tuples and relational:**

#### **Tuple:**

Each row in a table represents a record and is called a tuple. A table containing ‘n’ attributes in a record is called an n-tuple.

#### **Attributes:**

The name of each column in a table is used to interpret its meaning and is called an attribute. Each table is called a relation.

In the above table, account\_number, branch name, balance are the attributes.

#### **Domain:**

A domain is a set of values that can be given to an attribute. So every attribute in a table has a specific domain. Values to these attributes can not be assigned outside their domains.

#### **Relation:**

A relation consists of

- **Relational schema**
- **Relation instance**

#### **Relational schema:**

A relational schema specifies the relation’s name, its attributes and the domain of each attribute. If R is the name of a relation and A1,A2,... and is a list of attributes representing R then R(A1,A2,...,an) is called a relational schema. Each attribute in this relational schema takes a value from some specific domain called domain(Ai).

#### **Example:**

PERSON(PERSON\_ID:integer,NAME: STRING,AGE:INTEGER,ADDRESS:string)  
Total number of attributes in a relation denotes the degree of a relation. Since the PERSON relation schema contains four attributes, so this relation is of degree 4.

**Relation Instance:**

A relational instance denoted as  $r$  is a collection of tuples for a given relational schema at a specific point of time.

A relation state  $r$  to the relations schema  $R(A_1, A_2, \dots, A_n)$  also denoted by  $r^{\text{R}}$  is a set of  $n$ -tuples

$R\{t_1, t_2, \dots, t_m\}$

Where each  $n$ -tuple is an ordered list of  $n$  values

$T = \langle v_1, v_2, \dots, v_n \rangle$

Where each  $v_i$  belongs to domain  $(A_i)$  or contains null values.

The relation schema is also called 'intension' and the relation state is also called 'extension'.

Eg:

**Relation schema for student:**

STUDENT(rollno:string,name:string,city:string,age:integer)

**Relation instance:****Student:**

Rollno	Name	City	Age
101	Sujit	Bam	23
102	kunal	bbsr	22

**Keys:****Super key:**

A super key is an attribute or a set of attributes used to identify the records uniquely in a relation.

For example, customer-id, (cname, customer-id), (cname, telno)

**Candidate key:**

Super keys of a relation can contain extra attributes. candidate keys are minimal super keys. i.e, such a key contains no extraneous attribute. An attribute is called extraneous if even after removing it from the key, makes the remaining attributes still has the properties of a key.

In a relation  $R$ , a candidate key for  $R$  is a subset of the set of attributes of  $R$ , which have the following properties:

- *Uniqueness:* no two distinct tuples in  $R$  have the same values for the candidate key
- *Irreducible:* No proper subset of the candidate key has the *uniqueness property that is the candidate key.*
- *A candidate key's values must exist. It can't be null.*
- *The values of a candidate key must be stable. Its value can not change outside the control of the system.*

Eg: (cname, telno)

**Primary key:**

The primary key is the candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. The remaining candidate



keys if any are called *alternate key*.

### **RELATIONAL CONSTRAINTS:**

There are three types of constraints on relational database that include

- DOMAIN CONSTRAINTS
- KEY CONSTRAINTS
- INTEGRITY CONSTRAINTS

### **DOMAIN CONSTRAINTS:**

It specifies that each attribute in a relation an atomic value from the corresponding domains. The data types associated with commercial RDBMS domains include:

- Standard numeric data types for integer
- Real numbers
- Characters
- Fixed length strings and variable length strings

Thus, domain constraints specifies the condition that we to put on each instance of the relation. So the values that appear in each column must be drawn from the domain associated with that column.

Rollno	Name	City	Age
101	Sujit	Bam	23
102	kunal	Bbsr	22

### **Key constraints:**

This constraints states that the key attribute value in each tuple msut be unique .i.e, no two tuples contain the same value for the key attribute.(null values can allowed)

Emp(empcode,name,address) . here empcode can be unique

### **Integrity constraints:**

There are two types of integrity constraints:

- Entity integrity constraints
- Referential integrity constraints

### **Entity integrity constraints:**

It states that no primary key value can be null and unique. This is because the primary key is used to identify individual tuple in the relation. So we will not be able to identify the records uniquely containing null values for the primary key attributes. This constraint is specified on one individual relation.

### **Referential integrity constraints:**

It states that the tuple in one relation that refers to another relation must refer to an existing tuple in that relation. This constraints is specified on two relations .

If a column is declared as foreign key that must be primary key of another table.

**Department(deptcode,dname)**

Here the deptcode is the primary key.

**Emp(empcode,name,city,deptcode).**

Here the deptcode is foreign key.

**CODD'S RULES****Rule 1 : The information Rule.**

"All information in a relational data base is represented explicitly at the logical level and in exactly one way - by values in tables."

Everything within the database exists in tables and is accessed via table access routines.

**Rule 2 : Guaranteed access Rule.**

"Each and every datum (atomic value) in a relational data base is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name."

To access any data-item you specify which column within which table it exists, there is no reading of characters 10 to 20 of a 255 byte string.

**Rule 3 : Systematic treatment of null values.**

"Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type."

If data does not exist or does not apply then a value of NULL is applied, this is understood by the RDBMS as meaning non-applicable data.

**Rule 4 : Dynamic on-line catalog based on the relational model.**

"The data base description is represented at the logical level in the same way as-ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data."

The Data Dictionary is held within the RDBMS, thus there is no-need for off-line volumes to tell you the structure of the database.

**Rule 5 : Comprehensive data sub-language Rule.**

"A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all the following items

- Data Definition
- View Definition
- Data Manipulation (Interactive and by program).
- Integrity Constraints
- Authorization.

Every RDBMS should provide a language to allow the user to query the contents of the RDBMS and also manipulate the contents of the RDBMS.

**Rule 6 : .View updating Rule**

"All views that are theoretically updateable are also updateable by the system."

Not only can the user modify data, but so can the RDBMS when the user is not logged-in.

**Rule 7 : High-level insert, update and delete.**

"The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data."

The user should be able to modify several tables by modifying the view to which they act as base tables.

**Rule 8 : Physical data independence.**

"Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods."

The user should not be aware of where or upon which media data-files are stored

**Rule 9 : Logical data independence.**

"Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit un-impairment are made to the base tables."

User programs and the user should not be aware of any changes to the structure of the tables (such as the addition of extra columns).

**Rule 10 : Integrity independence.**

"Integrity constraints specific to a particular relational data base must be definable in the relational data sub-language and storable in the catalog, not in the application programs."

If a column only accepts certain values, then it is the RDBMS which enforces these constraints and not the user program, this means that an invalid value can never be entered into this column, whilst if the constraints were enforced via programs there is always a chance that a buggy program might allow incorrect values into the system.

**Rule 11 : Distribution independence.**

"A relational DBMS has distribution independence."

The RDBMS may spread across more than one system and across several networks, however to the end-user the tables should appear no different to those that are local.

**Rule 12 : Non-subversion Rule.**

"If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity Rules and constraints expressed in the higher level relational language (multiple-records-at-a-time)."

## **RELATION ALGEBRA:**

Relational algebra is a set of basic operations used to manipulate the data in relational model. These operations enable the user to specify basic retrieval request. The result of retrieval is anew relation, formed from one or more relation. These operation can be classified in two categories.

### ❖ **Basic Set Operation**

- Union
- Intersection
- Set difference
- Cartesian product

### ❖ **Relational operations**

- Select
- Project
- Join
- Division

### **Basic set operation:**

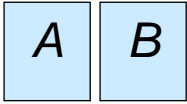
These are the binary operations; i.e, each is applied to two sets or relations. These two relations should be union compatible except in case of Cartesian product.

Two relations  $R(A_1, A_2, \dots, A_n)$  and  $S(B_1, B_2, \dots, B_n)$  are said to be union compatible if they have the same degree  $n$  and domains of the corresponding attributes are also the same;  $\text{domain}(A_i) = \text{Domain}(B_i)$  for  $1 \leq i \leq n$ .

## Union Operation – Example

Relations  $r, s$ :

$r \cup s$ :



A	B
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

A	B
$\alpha$	2
$\beta$	3

$s$

$\alpha$	1
$\alpha$	2
$\beta$	1
$\beta$	3

### UNION OPERATION:

Notation:  $r \cup s$

Defined as:

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

For  $r \cup s$  to be valid.

$r, s$  must have the *same arity* (same number of attributes)

2. The attribute domains must be *compatible* (e.g., 2nd column of  $r$  deals with the same type of values as does the 2nd column of  $s$ )

E.g. to find all customers with either an account or a loan  $\Pi_{customer-name} (depositor) \cup \Pi_{customer-name} (borrower)$

**Operation – Example**  
Relations  $r, s$ :

**Set Difference**

$r - s$ :

A	B

A	B
$\alpha$	1
$\alpha$	2
$\beta$	1

*r*

A	B
$\alpha$	2
$\beta$	3

*s*

$\alpha$	1
$\beta$	1

### Set Difference Operation

- Notation  $r - s$
- Defined as:
  - $r - s = \{t \mid t \in r \text{ and } t \notin s\}$
- Set differences must be taken between *compatible* relations.
  - $r$  and  $s$  must have the *same arity*
  - attribute domains of  $r$  and  $s$  must be compatible

# Cartesian product – Example

A	B
$\alpha$	1
$\beta$	2

*r*

C	D	E
$\alpha$	10	a
$\beta$	10	a
$\beta$	20	b
$\gamma$	10	b

*s*

A	B	C	D	E
$\alpha$	1	$\alpha$	10	a
$\alpha$	1	$\beta$	10	a
$\alpha$	1	$\beta$	20	b
$\alpha$	1	$\gamma$	10	b
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b
$\beta$	2	$\gamma$	10	b

# Project Operation – Example

Relation *r*:

A	B	C
$\alpha$	10	1
$\alpha$	20	1
$\beta$	30	1
$\beta$	40	2

*r* X *s*:

Notation *r* x *s*

Defined as:

$\Pi_{A,C}(r) = \{t \mid t \in r \text{ and } q \in s\}$

Assume that attributes of *r* and *s* are disjoint. (That is,  $R \cap S = \emptyset$ )

If attributes of *r* and *s* are not disjoint, then renaming must be used. where *A*<sub>1</sub>, *A*<sub>2</sub>, ..., *A*<sub>*k*</sub> are attribute names and *r*<sub>1</sub>, *r*<sub>2</sub>, ..., *r*<sub>*k*</sub> are relation names.

The result is defined as the relation of *k* columns obtained by erasing the columns that are not listed

Duplicate rows removed from result, since relations are sets

E.g. To eliminate the *branch-name* attribute of *account*

$\Pi_{\text{account-number, balance}}(\text{account})$



## Select Operation – Example

- Relation  $r$

A	B	C	D
$\alpha$	$\alpha$	1	7
$\alpha$	$\beta$	5	7
$\beta$	$\beta$	12	3
$\beta$	$\beta$	23	10

- $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
$\alpha$	$\alpha$	1	7
$\beta$	$\beta$	23	10

Notation:  $\sigma_p(r)$

$p$  is called the **selection predicate**

Defined as:

$$\sigma_p(r) = \{t \mid t \in r \text{ and } p(t)\}$$

Where  $p$  is a formula in propositional calculus consisting of **terms** connected by :  $\wedge$  (and),  $\vee$  (or),  $\neg$  (not) Each **term** is one of:

<attribute>  $op$  <attribute> or <constant>

where  $op$  is one of: =,  $\neq$ ,  $>$ ,  $\geq$ ,  $<$ ,  $\leq$

Example of selection:

**Q:** Display the account details belonging to the branch “perryridge”.  $\sigma_{branch-name="Perryridge"}(account)$

## Rename Operation

Allows us to name, and therefore to refer to, the results of relational-algebra expressions.

Allows us to refer to a relation by more than one name.

Example:

$$\rho_X(E)$$

returns the expression  $E$  under the name  $X$

If a relational-algebra expression  $E$  has arity  $n$ , then

$$\rho_X(A_1, A_2, \dots, A_n)(E)$$

returns the result of expression  $E$  under the name  $X$ , and with the attributes renamed to  $A_1, A_2, \dots, A_n$ .

## Set-Intersection Operation

- Notation:  $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$
- Assume:
  - $r, s$  have the *same arity*
  - attributes of  $r$  and  $s$  are compatible
- Note:  $r \cap s = r - (r - s)$

### Set-Intersection Operation - Example

$n$  Relation  $r, s$ :

A	B
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

A	B
$\alpha$	2
$\beta$	3

$s$

$n$   $r \cap s$

A	B
$\alpha$	2

# Natural-Join Operation

□ Notation:  $r$  and  $s$  are relations on schemas  $R$  and  $S$  respectively.  
 □ Then,  $r \bowtie s$  is a relation on schema  $R \cup S$  obtained as follows:

- Consider each pair of tuples  $t_r$  from  $r$  and  $t_s$  from  $s$ .
- If  $t_r$  and  $t_s$  have the same value on each of the attributes in  $R \cap S$ , add a tuple  $t$  to the result, where
  - $t$  has the same value as  $t_r$  on  $r$
  - $t$  has the same value as  $t_s$  on  $s$

□ Example:

$R = (A, B, C, D)$

$S = (E, B, D)$

□ Result schema =  $(A, B, C, D, E)$

□  $r \bowtie s$  is defined as:

$$\Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$$

## Natural Join Operation – Example

□ Relations  $r, s$ :

A	B	C	D
$\alpha$	1	$\alpha$	a
$\beta$	2	$\gamma$	a
$\gamma$	4	$\beta$	b
$\alpha$	1	$\gamma$	a
$\delta$	2	$\beta$	b

$r$

B	D	E
1	a	$\alpha$
3	a	$\beta$
1	a	$\gamma$
2	b	$\delta$
3	b	$\epsilon$

$s$

$r \bowtie s$

A	B	C	D	E
$\alpha$	1	$\alpha$	a	$\alpha$
$\alpha$	1	$\alpha$	a	$\gamma$
$\alpha$	1	$\gamma$	a	$\alpha$
$\alpha$	1	$\gamma$	a	$\gamma$
$\delta$	2	$\beta$	b	$\delta$

□

# Division Operation

$$r \div s$$

Suited to queries that include the phrase “for all”.  
 Let  $r$  and  $s$  be relations on schemas  $R$  and  $S$   
 respectively where

$$R = (A_1, \dots, A_m, B_1, \dots, B_n)$$

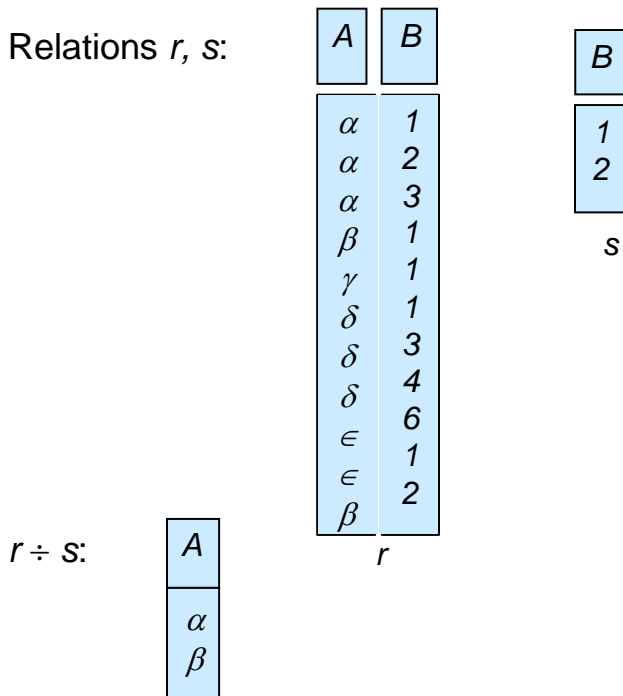
$$S = (B_1, \dots, B_n)$$

The result of  $r \div s$  is a relation on schema

$$R - S = (A_1, \dots, A_m)$$

$$r \div s = \{ t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$$

## Division Operation – Example





## Example Queries

- n Find all customers who have an account from at least the “Downtown” and the Uptown” branches.

Query 1

$$\Pi_{CN}(\sigma_{BN=\text{“Downtown”}}(depositor \bowtie account)) \cap \\ \Pi_{CN}(\sigma_{BN=\text{“Uptown”}}(depositor \bowtie account))$$

where **CN** denotes customer-name and **BN** denotes **branch-name**.

Query 2

$$\Pi_{customer-name, branch-name}(depositor \bowtie account) \\ \div \rho_{temp(branch-name)}(\{\text{“Downtown”}, \text{“Uptown”}\})$$

## Example Queries

- n Find all customers who have an account at all branches located in Brooklyn city.

$$\Pi_{customer-name, branch-name}(depositor \bowtie account) \\ \div \Pi_{branch-name}(\sigma_{branch-city = \text{“Brooklyn”}}(branch))$$

## Banking Example

*branch* (*branch-name*, *branch-city*, *assets*)

*customer* (*customer-name*, *customer-street*, *customer-only*)

*account* (*account-number*, *branch-name*, *balance*)

*loan* (*loan-number*, *branch-name*, *amount*)

*depositor* (*customer-name*, *account-number*)

*borrower* (*customer-name*, *loan-number*)

## Example Queries

- n Find all loans of over \$1200

$$\sigma_{amount > 1200} (loan)$$

- n Find the loan number for each loan of an amount greater than \$1200

$$\Pi_{loan-number} (\sigma_{amount > 1200} (loan))$$

## Example Queries

- n Find the names of all customers who have a loan, an account, or both, from the bank

$$\Pi_{customer-name} (borrower) \cup \Pi_{customer-name} (depositor)$$

- n Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer-name} (borrower) \cap \Pi_{customer-name} (depositor)$$

## Example Queries

Find the names of all customers who have a loan at the Perryridge branch.

$$\Pi_{customer-name} (\sigma_{branch-name="Perryridge"} (\sigma_{borrower.loan-number = loan.loan-number}(borrower \times loan)))$$

Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$$\Pi_{customer-name} (\sigma_{branch-name = "Perryridge"} (\sigma_{borrower.loan-number = loan.loan-number}(borrower \times loan))) - \Pi_{customer-name}(\text{depositor})$$

## Example Queries

n Find the names of all customers who have a loan at the Perryridge branch.

– Query 1

$$\Pi_{customer-name} (\sigma_{branch-name = "Perryridge"} (\sigma_{borrower.loan-number = loan.loan-number}(borrower \times loan)))$$

– Query 2

$$\Pi_{customer-name} (\sigma_{loan.loan-number = borrower.loan-number} (\sigma_{branch-name = "Perryridge"}(loan)) \times borrower)$$

## Example Queries

Find the largest account balance

n Rename *account* relation as *d*

n The query is:

$$\Pi_{balance}(\text{account}) - \Pi_{account.balance} (\sigma_{account.balance < d.balance} (\text{account} \times \rho_d(\text{account})))$$



## Aggregate functions:

**Aggregation function** takes a collection of values and returns a single value as a result.

**avg**: average value  
**min**: minimum value  
**max**: maximum value  
**sum**: sum of values  
**count**: number of values

**Aggregate operation** in relational algebra

$G_1, G_2, \dots, G_n \ g \ F_1(A_1), F_2(A_2), \dots,$

$F_n(A_n) (E)$

$E$  is any relational-algebra expression

$G_1, G_2, \dots, G_n$  is a list of attributes on which to group (can be empty)

Each  $F_i$  is an aggregate function.

Each  $A_i$  is an attribute name

Find sum of sal for all emp records

$G_{sum(sal)}(emp)$

Find maximum salary from emp table

$G_{max(salary)}(emp)$

Find branch name and maximum salary from emp table

$Branch\_name \ G_{max(salary)}(emp)$

## Aggregate Operation – Example

n Relation  $r$ :

A	B	C
$\alpha$	$\alpha$	7
$\alpha$	$\beta$	7
$\beta$	$\beta$	3
$\beta$	$\beta$	10

$g_{sum(c)}(r)$

sum-C
27

## OUTER JOIN:

The outer join operation is an extension of the join operation to deal with missing information.

There are three forms of outer join

- left outer join
- right outer join
- full outer join

**employee:**

Empname	Street	City
Coyote	Toon	Hollywood
Rabbit	Tunnel	carrot
Smith	Revolver	Death valley
William	Seaview	Seattle

**ft\_works:**

Empname	Branch name	Salary
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
William	Redmond	1500

Employee  $\bowtie$  ft\_works

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500

## Left outer join:

It takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes. The right relation and adds them to the result of the natural join. In tuple (smith, Revolver, Death valley, null, null) is such a tuple. All information from the left relation is present in the result of the left outer join.

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death valley	Null	null

Result of Employee  ft\_works

**Right outer join:**


It is symmetric with the left outer join. It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join. tuple(Gates,null,null,Redmond,5300) is such a tuple. Thus, all information from the right relation is present in the result of the right outer join.

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500
gates	Null	Null	Redmond	5300

**Full outer join:**

It does both of those operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join. Figure 3.35 shows the result of a full outer join.

Since outer join operations may generate results containing null values, we need to specify how the different relation-algebra operations deal with null values. It is interesting to note that the outer join operations can be expressed by the basic relational algebra operations. For instance the left outer join operation

Employee  ft\_works

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500
gates	Null	Null	Redmond	5300

## Tuple Relational Calculus

The tuple relational calculus is a non procedural query language. It describes the desired information with out giving a specific procedure for obtaining that information.

A query in the tuple relational calculus is expressed as:

$$\{t|P(t)\}$$

where P is a formula. Several tuple variables may appear in a formula. A tuple variable is said to be a free variable unless it is quantified by a  $\exists$  or  $\forall$ .

n A nonprocedural query language, where each query is of the form

## Example Queries

n Find the *loan-number*, *branch-name*, and *amount* for loans of over \$1200

$$\{t \mid t \in loan \wedge t[amount] > 1200\}$$

n Find the loan number for each loan of an amount greater than \$1200

$$\{t \mid \exists s \in loan (t[loan-number] = s[loan-number] \wedge s[amount] > 1200)\}$$

Notice that a relation on schema [*loan-number*] is implicitly defined by the query

## Example Queries

Find the names of all customers having a loan, an account, or both at the bank

Find the names of all customers having a loan, an account, or both at the bank

$$\{t \mid \exists s \in borrower (t[customer-name] = s[customer-name]) \vee \exists u \in depositor (t[customer-name] = u[customer-name])\}$$

$$\{t \mid \exists s \in borrower (t[customer-name] = s[customer-name]) \wedge \exists u \in loan (u[branch-name] = "Perryridge" \wedge t[customer-name] = u[customer-name])\}$$

Find the names of all customers who have a loan and an account at the bank

## Safety of Expressions

Find the names of all customers who have a loan and an account at the bank

It is possible to write tuple calculus expressions that generate infinite relations

$$\{t \mid \exists s \in borrower (t[customer-name] = s[customer-name]) \wedge \exists u \in depositor (t[customer-name] = u[customer-name])\}$$

# Domain Relational Calculus

A nonprocedural query language equivalent in power to the tuple relational calculus

Each query is an expression of the form:

Find the *loan-number*, *branch-name*, and *amount* for loans of

## Query-by-Example (QBE)

QBE — Basic Structure

- n A graphical query language which is based (roughly) on the domain relational calculus
- n Two dimensional syntax – system creates templates of relations that are requested by users
- n Queries are expressed “by example”

## QBE Skeleton Tables for the Bank Example

<i>branch</i>	<i>branch-name</i>	<i>branch-city</i>	<i>assets</i>

<i>customer</i>	<i>customer-name</i>	<i>customer-street</i>	<i>customer-city</i>

<i>loan</i>	<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>

## QESkeleton Tables (Cont.)

<i>borrower</i>	<i>customer-name</i>	<i>loan-number</i>

<i>account</i>	<i>account-number</i>	<i>branch-name</i>	<i>balance</i>

<i>depositor</i>	<i>customer-name</i>	<i>account-number</i>

## Queries on One Relation

- Find all loan numbers at the Perryridge branch.

<i>loan</i>	<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>
	P._x	Perryridge	

- \_x is a variable (optional; can be omitted in above query)
- P. means print (display)
- duplicates are removed by default
- To retain duplicates use P.ALL

<i>loan</i>	<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>
	P.ALL.	Perryridge	

## Queries on One Relation (Cont.)

- Display full details of all loans

Method 1:

<i>loan</i>	<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>
	P_x	P_y	P_z

Method 2: Shorter notation

<i>loan</i>	<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>
P.			

## Queries on One Relation (Cont.)

- Find the loan number of all loans with a loan amount of more than \$700

<i>loan</i>	<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>
	P.		>700

- Find names of all branches that are not located in Brooklyn

<i>branch</i>	<i>branch-name</i>	<i>branch-city</i>	<i>assets</i>
	P.	¬ Brooklyn	

## Queries on One Relation (Cont.)

- Find the loan numbers of all loans made jointly to Smith and Jones

<i>borrower</i>	<i>customer-name</i>	<i>loan-number</i>
	"Smith"	P. <sub>x</sub>
	"Jones"	<sub>x</sub>

- Find all customers who live in the same city as Jones

<i>customer</i>	<i>customer-name</i>	<i>customer-street</i>	<i>customer-city</i>
	P. <sub>x</sub>		<sub>y</sub>
	Jones		<sub>y</sub>

## Queries on Several Relations

- Find the names of all customers who have a loan from the Perryridge branch

<i>loan</i>	<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>
	<sub>x</sub>	Perryridge	
<i>borrower</i>	<i>customer-name</i>	<i>loan-number</i>	
	P. <sub>y</sub>	<sub>x</sub>	



## Queries on Several Relations (Cont)

- Find the names of all customers who have both an account and a loan at the bank

<i>depositor</i>	<i>customer-name</i>	<i>account-number</i>
	P..x	

<i>borrower</i>	<i>customer-name</i>	<i>loan-number</i>
	_x	

## Negation in QBE

- Find the names of all customers who have an account at the bank, but do not have a loan from the bank

<i>depositor</i>	<i>customer-name</i>	<i>account-number</i>
	P..x	

<i>borrower</i>	<i>customer-name</i>	<i>loan-number</i>
$\neg$	_x	

## Negation in QBE (Cont.)

- Find all customers who have at least two accounts

<i>depositor</i>	<i>customer-name</i>	<i>account-number</i>
	P. <i>x</i>	<i>y</i>
	<i>x</i>	$\neg$ <i>y</i>

$\neg$  means "not equal to"

## The Condition Box

- Allows the expression of constraints on domain variables that are either inconvenient or impossible to express within the skeleton tables.
- Complex conditions can be used in condition boxes
- Eg. Find the loan numbers of all loans made to Smith, to Jones, or to both jointly

<i>borrower</i>	<i>customer-name</i>	<i>loan-number</i>		
	<i>n</i>	P. <i>x</i>		
<table border="1" style="margin: auto;"> <tr> <td><i>conditions</i></td> </tr> <tr> <td><i>n</i> = Smith or <i>n</i> = Jones</td> </tr> </table>			<i>conditions</i>	<i>n</i> = Smith or <i>n</i> = Jones
<i>conditions</i>				
<i>n</i> = Smith or <i>n</i> = Jones				

## Condition Box (Cont.)

- ? QBE supports an interesting syntax for expressing alternative values

<i>branch</i>	<i>branch-name</i>	<i>branch-city</i>	<i>assets</i>		
	P.	<i>_x</i>			
<table border="1"> <tr> <td><i>conditions</i></td> </tr> <tr> <td><i>_x</i> = (Brooklyn or Queens)</td> </tr> </table>				<i>conditions</i>	<i>_x</i> = (Brooklyn or Queens)
<i>conditions</i>					
<i>_x</i> = (Brooklyn or Queens)					

## Condition Box (Cont.)

- Find all account numbers with a balance between \$1,300 and \$1,500

<i>account</i>	<i>account-number</i>	<i>branch-name</i>	<i>balance</i>			
	P.		<i>_x</i>			
<table border="1"> <tr> <td><i>conditions</i></td> </tr> <tr> <td><i>_x</i> ≥ 1300</td> </tr> <tr> <td><i>_x</i> ≤ 1500</td> </tr> </table>				<i>conditions</i>	<i>_x</i> ≥ 1300	<i>_x</i> ≤ 1500
<i>conditions</i>						
<i>_x</i> ≥ 1300						
<i>_x</i> ≤ 1500						

- Find all account numbers with a balance between \$1,300 and \$2,000 but not exactly \$1,500.

## The Result Relation

- n Find the *customer-name*, *account-number*, and *balance* for all customers who have an account at the Perryridge branch.

H We need to:

- 4 Join *depositor* and *account*.

- 4 Project *customer-name*, *account-number* and *balance*.

H To accomplish this we:

- 4 Create a skeleton table, called *result*, with attributes *customer-name*, *account-number*, and *balance*.

- 4 Write the query.

## The Result Relation (Cont.)

■ The resulting query is

<i>account</i>	<i>account-number</i>	<i>branch-name</i>	<i>balance</i>
	<i>-y</i>	Perryridge	<i>-z</i>

<i>depositor</i>	<i>customer-name</i>	<i>account-number</i>
	<i>-x</i>	<i>-y</i>

## Ordering the Display of Tuples

- AO= ascending order, DO= descending order.
- Eg. list in ascending alphabetical order all customers who have an account at the bank

<i>depositor</i>	<i>customer-name</i>	<i>account-number</i>
	P.AO.	

- When sorting on multiple attributes, the sorting order is specified by including with each sort operator (AO or DO) an integer surrounded by parentheses.
- Eg. List all account numbers at the Perryridge branch in ascending alphabetic order with their respective account balances in descending order.

<i>account</i>	<i>account-number</i>	<i>branch-name</i>	<i>balance</i>
	P.AO(1).	Perryridge	P.DO(2).

## Aggregate Operations

- The aggregate operators are AVG, MAX, MIN, SUM, and CNT
- The above operators must be postfix with "ALL" (eg,

## Aggregate Operations (Cont.)

- n UNQ is used to specify that we want to eliminate duplicates
- n Find the total number of customers having an account at the bank.

<i>depositor</i>	<i>customer-name</i>	<i>account-number</i>
	P.CNT.UNQ.	

## Modification of the Database – Deletion

- n Deletion of tuples from a relation is expressed by use of a D. command. In the case where we delete information in only some of the columns, null values, specified by –, are inserted.
- n Delete customer Smith

<i>customer</i>	<i>customer-name</i>	<i>customer-street</i>	<i>customer-city</i>
D.	Smith		

- n Delete the *branch-city* value of the branch whose name is “Perryridge”.

<i>branch</i>	<i>branch-name</i>	<i>branch-city</i>	<i>assets</i>
	Perryridge	D.	

## Deletion Query Examples

- n Delete all loans with a loan amount between \$1300 and \$1500.
  - H For consistency, we have to delete information from loan and borrower tables

<i>loan</i>	<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>
D.	<i>-y</i>		<i>-x</i>

## Modification of the Database – Insertion

- n Insertion is done by placing the I. operator in the query expression.
- n Insert the fact that account A-9732 at the Perryridge branch has a balance of \$700.

<i>account</i>	<i>account-number</i>	<i>branch-name</i>	<i>balance</i>
I.	A-9732	Perryridge	700

## Modification of the Database – Updates

Use the U. operator to change a value in a tuple without changing *all* values in the tuple. QBE does not allow users to update the primary key fields.

Update the asset value of the Perryridge branch to \$10,000,000.

BRANCH	BRANCH-TYPE	BRANCH-CITY	ASSETS
Perryridge	BR	Perryridge	10,000,000

BRANCH	BRANCH-TYPE	BRANCH-CITY	ASSETS
Perryridge	BR	Perryridge	10,000,000

RELATIONAL DATABASE DEGIN



Data base design is a process in which you create a logical data model for a database, which store data of a company. It is performed after initial database study phase in the database life cycle. You use normalization technique to create the logical data model for a database and eliminate data redundancy. Normalization also allows you to organize data efficiently in a data base and reduce anomalies during data operation. Various normal forms, such as first, second and third can be applied to create a logical data model for a database. The second and third normal forms are based on partial dependency and transitivity dependency. Partial dependency occurs when a row of table is uniquely identified by one column that is a part of a primary key. A transitivity dependency ours when a non key column is uniquely identified by values in another non-key column of a table.

### **Data base design process:**

We can identify six main phases of the database design process:

1. Requirement collection and analysis
2. Conceptual data base design
3. Choice of a DBMS
4. Data model mapping(logical database design)
5. physical data base design
6. database system implementation and tuning

#### **1. Requirement collection and analysis**

Before we can effectively design a data base we must know and analyze the expectation of the users and the intended uses of the database in as much as detail.

#### **2. Conceptual data base design**

The goal for this phase I s to produce a conceptual schema for the database that is independent of a specific DBMS.

- We often use a high level data model such er-model during this phase
- We specify as many of known database application on transactions as possible using a notation the is independent of any specific dbms.
- Often the dbms choice is already made for the organization the intent of conceptual design still to keep , it as free as possible from implementation consideration.

#### **3. Choice of a DBMS**

The choice of dbms is governed by a no. of factors some technical other economic and still other concerned with the politics of the organization.

The economics and organizational factors that offer the choice of the dbms are: Software cost, maintenance cost, hardware cost, database creation and conversion cost, personnel cost, training cost, operating cost.

4. **Data model mapping (logical database design)**

During this phase, we map the conceptual schema from the high level data model used on phase 2 into a data model of the choice dbms.

5. **Physical database design**

During this phase we design the specification for the database in terms of physical storage structure ,record placement and indexes.

6. **Database system implementation and tuning**

During this phase, the database and application programs are implemented, tested and eventually deployed for service.

## UNIT-3

### SCHEMA REFINEMENT AND NORMALISATION

#### Unit 3 contents at a glance:

1. Introduction to schema refinement,
2. functional dependencies,
3. reasoning about FDs.
4. Normal forms: 1NF, 2NF, 3NF, BCNF,
5. properties of decompositions,
6. normalization,
7. schema refinement in database design(Refer Text Book),
8. other kinds of dependencies: 4NF, 5NF, DKNF
9. Case Studies(Refer text book)

#### 1. Schema Refinement:

The Schema Refinement refers to refine the schema by using some technique. The best technique of schema refinement is **decomposition**.

***Normalisation or Schema Refinement is a technique of organizing the data in the database. It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies.***

***Redundancy*** refers to repetition of same data or duplicate copies of same data stored in different locations.

***Anomalies:*** Anomalies refers to the problems occurred after poorly planned and normalised databases where all the data is stored in one table which is sometimes called a flat file database.

**Anomalies or problems facing without normalization(problems due to redundancy) :**

Anomalies refers to the problems occurred after poorly planned and unnormalised databases where all the data is stored in one table which is sometimes called a flat file database. Let us consider such type of schema –

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k
<b>Primary Key(SID,CID)</b>				

Here all the data is stored in a single table which causes redundancy of data or say anomalies as SID and Sname are repeated once for same CID . Let us discuss anomalies one by one.

Due to redundancy of data we may get the following problems, those are-

**1.insertion anomalies :** It may not be possible to store some information unless some other information is stored as well.

**2.redundant storage:** some information is stored repeatedly

**3.update anomalies:** If one copy of redundant data is updated, then inconsistency is created unless all redundant copies of data are updated.

**4.deletion anomalies:** It may not be possible to delete some information without losing some other information as well.

**Problem in updation / updation anomaly** – If there is updation in the fee from 5000 to 7000, then we have to update FEE column in all the rows, else data will become inconsistent.

SID	Sname	CID	Cname	FEE
S1	A	C1	C	<del>5k</del>
S2	A	C1	C	<del>5k</del>
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k



**Insertion Anomaly and Deletion Anomaly-** These anomalies exist only due to redundancy, otherwise they do not exist.

**Insertion Anomalies:** New course is introduced C4, But no student is there who is having C4 subject.

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

NULL	NULL	CA	DB	12k
------	------	----	----	-----

To Insert that Row, It is Required to Put Dummy Data..

Therefore,

xx	xx	CA	DB	12k
----	----	----	----	-----

Because of insertion of some data, It is forced to insert some other dummy data.

**Deletion Anomaly :**

Deletion of S3 student cause the deletion of course.

Because of deletion of some data forced to delete some other useful data.

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
<del>S3</del>	<del>B</del>	<del>C2</del>	<del>C</del>	<del>10k</del>
<del>S3</del>	<del>B</del>	<del>C2</del>	<del>JAVA</del>	<del>15k</del>

**Solutions To Anomalies : Decomposition of Tables – Schema Refinement**

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

SID	Sname	CID
S1	A	C1
S2	A	C1
S1	A	C2
<del>S3</del>	<del>B</del>	<del>C2</del>
<del>S3</del>	<del>B</del>	<del>C2</del>

PK(SID,CID)

Deletion Anamoly Removed

CID	CNAME	FEE
C1	C	<del>5k</del>
C2	C	10k
C3	JAVA	15k
C4	DB	12k

PK(CID)

7k (Updation Anamoly Removed)

Insertion Anamoly Removed

There are some Anomalies in this again –

SID	Sname	CID
S1	<del>A</del> (AA)	C1
S2	<del>A</del> (AA)	C1
S1	<del>A</del> (AA)	C2
S3	B	C2
S3	B	C3
S4	B	xx

Update Anamoly

Deletion Anamoly as  
C2 course is allotted  
to some students

CID	CNAME	FEE
C1	C	5k
C2	C	10k
C3	JAVA	15k
C4	DB	12k

A student having no course is enrolled. We have to put dummy data again.

**What is the Solution ??**

**Solution : decomposing into relations as shown below**

R1

SID	Sname

R2

SID	CID

R3

CID	Cname	Fee

→ **TO AVOID REDUNDANCY** and problems due to redundancy, we use refinement technique called **DECOMPOSITION**.

**Decomposition:-** Process of decomposing a larger relation into smaller relations.

→ Each of smaller relations contain subset of attributes of original relation.

**Functional dependencies:**

→ Functional dependency is a relationship that exist when one attribute uniquely determines another attribute.

→ Functional dependency is a form of integrity constraint that can identify schema with redundant storage problems and to suggest refinement.

→ A functional dependency  $A \rightarrow B$  in a relation holds true if two tuples having the same value of attribute A also have the same value of attribute B

**IF  $t1.X=t2.X$  then  $t1.Y=t2.Y$**  where t1,t2 are tuples and X,Y are attributes.

**Reasoning about functional dependencies:****Armstrong Axioms :**

Armstrong axioms defines the set of rules for reasoning about functional dependencies and also to infer all the functional dependencies on a relational database.

**Various axioms rules or inference rules:**

Primary axioms:

<b>Rule 1</b>	<b>Reflexivity</b> If A is a set of attributes and B is a subset of A, then A holds B. $\{ A \rightarrow B \}$
<b>Rule 2</b>	<b>Augmentation</b> If A hold B and C is a set of attributes, then AC holds BC. $\{AC \rightarrow BC\}$ It means that attribute in dependencies does not change the basic dependencies.
<b>Rule 3</b>	<b>Transitivity</b> If A holds B and B holds C, then A holds C. If $\{A \rightarrow B\}$ and $\{B \rightarrow C\}$ , then $\{A \rightarrow C\}$ A holds B $\{A \rightarrow B\}$ means that A functionally determines B.

secondary or derived axioms:

<b>Rule 1</b>	<b>Union</b> If A holds B and A holds C, then A holds BC. If $\{A \rightarrow B\}$ and $\{A \rightarrow C\}$ , then $\{A \rightarrow BC\}$
<b>Rule 2</b>	<b>Decomposition</b> If A holds BC and A holds B, then A holds C. If $\{A \rightarrow BC\}$ and $\{A \rightarrow B\}$ , then $\{A \rightarrow C\}$
<b>Rule 3</b>	<b>Pseudo Transitivity</b> If A holds B and BC holds D, then AC holds D. If $\{A \rightarrow B\}$ and $\{BC \rightarrow D\}$ , then $\{AC \rightarrow D\}$

**Attribute closure:** Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.

NOTE:

To find attribute closure of an attribute set-

- 1)add elements of attribute set to the result set.
- 2)recursively add elements to the result set which can be functionally determined from the elements of result set.

*Algorithm : Determining  $X^+$ , the closure of X under F.*

**Input :** Let F be a set of FDs for relation R.

**Steps:**

```

1.  $X^+ = X$  //initialize  $X^+$  to X
2. For each FD :  $Y \rightarrow Z$  in F Do
   If  $Y \subseteq X^+$  Then //If Y is contained in  $X^+$ 
    $X^+ = X^+ \cup Z$  //add Z to  $X^+$ 
   End If
   End For
3. Return  $X^+$  //Return closure of X

```

**Output :** Closure  $X^+$  of X under F

**Types of functional dependencies:**

1)**Trivial functional dependency:**-If  $X \rightarrow Y$  is a functional dependency where  $Y \subseteq X$ , these type of FD's called as trivial functional dependency.

2)**Non-trivial functional dependency:**-If  $X \rightarrow Y$  and  $Y$  is not subset of  $X$  then it is called non-trivial functional dependency.

3)**Completely non-trivial functional dependency:**-If  $X \rightarrow Y$  and  $X \cap Y = \Phi$ (null) then it is called completely non-trivial functional dependency.

**Prime and non-prime attributes**

Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes.

**Candidate Key:**

Candidate Key is minimal set of attributes of a relation which can be used to identify a tuple uniquely.

Consider student table: student(sno, sname,sphone,age)

we can take **sno** as candidate key. we can have more than 1 candidate key in a table.

types of candidate keys:

1. simple(having only one attribute)
2. composite(having multiple attributes as candidate key)

**Super Key:**

Super Key is set of attributes of a relation which can be used to identify a tuple uniquely.

- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

Consider student table: student(sno, sname,sphone,age)

we can take sno, (sno, sname) as super key



**Finding candidate keys problems:**

**Example 1:** Find candidate keys for the relation R(ABCD) having following FD's

$AB \rightarrow CD, C \rightarrow A, D \rightarrow A$

**Solution:**

$AB^+ = \{ABCD\}$  A and B are prime attributes

$C \rightarrow A$  replace A by c

$BC^+ = \{ABCD\}$  A and C are prime attributes ( $A^+ = A^+ = \{AC\}$ )

$D \rightarrow B$  replace B by D

$AD^+ = \{ABCD\}$  A and D are prime attributes ( $D^+ = \{BD\}$ )

$CD^+ = \{ABCD\}$  (replacing A by C in AD)

AB, BC, CD, AD are candidate keys.

**Example 2:** Find candidate keys for R(ABCDE) having following FD's

$A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A$

**Solution:**

$A^+ = \{ABCDE\}$  A is candidate key and prime attribute

$E \rightarrow A$  so replace A by E

$E^+ = \{ABCDE\}$  E is candidate key and prime attribute

$CD \rightarrow E$  replace E by CD

$CD^+ = \{ABCDE\}$  ( $C^+ = C$  and  $D^+ = D$ ) no proper subset of CD is superkey. so CD is candidate key

$B \rightarrow D$

$BC^+ = \{ABCDE\}$  ( $B^+ = BD$ ) BC is candidate key

A, E, CD, BC are candidate keys

**Question 1 :** Given a relation R(ABCDEF) having FDs {AB→C, C→D, D→E, F→B, E→F} Identify the prime attributes and non prime attributes .

**Solution :**

$(AB)^+ : \{ABCDEF\} \Rightarrow$  Super Key

$(A)^+ : \{A\} \Rightarrow$  Not Super Key

$(B)^+ : \{B\} \Rightarrow$  Not Super Key

**Prime Attributes :** {A,B}

$(AB) \rightarrow$  Candidate Key

↓ (as  $F \rightarrow B$ )

$(AF)^+ : \{AFBCDE\}$

$(A)^+ : \{A\} \Rightarrow$  Not Super key

$(F)^+ : \{FB\} \Rightarrow$  Not Super Key

$(AF) \rightarrow$  Candidate Key

↓

$(AE)^+ : \{AEFBCD\}$

$(A)^+ : \{A\} \Rightarrow$  Not Super key

$(E)^+ : \{EFB\} \Rightarrow$  Not Super key

$(AE) \rightarrow$  Candidate Key

↓

$(AD)^+ : \{ADEFB\}$

$(A)^+ : \{A\} \Rightarrow$  Not Super key

$(D)^+ : \{DEFB\} \Rightarrow$  Not Super key

$(AD) \rightarrow$  Candidate Key

↓

$(AC)^+ : \{ACDEFB\}$

$(A)^+ : \{A\} \Rightarrow$  Not Super Key

$(C)^+ : \{DCEFB\} \Rightarrow$  Not Super Key

$\Rightarrow$  Candidate Keys {AB, AF, AE, AD, AC}

$\Rightarrow$  Prime Attributes {A,B,C,D,E,F}

$\Rightarrow$  Non Prime Attributes { }

**Question 2:** Given a relation R(ABCDEF) having FDs {AB → C, C → DE, E → F, C → B} Identify the prime attributes and non prime attributes.

**Solution :**

$(AB)^+ : \{A B C D E F\}$

$(A)^+ : \{A\}$

$(B)^+ : \{B\}$

$(AB) \Rightarrow (AC), (AC)^+ : \{ABCDEF\}$

$(C)^+ : \{DECBF\}$

$\Rightarrow$  Candidate Keys {AB, AC}

$\Rightarrow$  Prime Attributes {A,B,C}

$\Rightarrow$  Non Prime Attributes {D,E,F}

**Normalization:**

Normalization is a process of designing a consistent database with minimum redundancy which support data integrity by grating or decomposing given relation into smaller relations preserving constraints on the relation.

→Normalisation removes data redundancy and it will helps in designing a good data base which involves a set of normal forms as follows -

- 1)First normal form(1NF)
- 2)Second normal form(2NF)
- 3)Third normal form(3NF)
- 4)Boyce coded normal form(BCNF)
- 5)Forth normal form(4NF)
- 6)Fifth normal form(5NF)
- 7)Sixth normal form(6NF)
- 8)Domain key normal form.

## Normal Forms

1 <sup>st</sup> Normal Form	No repeating data groups
2 <sup>nd</sup> Normal Form	No partial key dependency
3 <sup>rd</sup> Normal Form	No transitive dependency
Boyce-Codd Normal Form	Reduce keys dependency
4 <sup>th</sup> Normal Form	No multi-valued dependency
5 <sup>th</sup> Normal Form	No join dependency

$$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$$

Property	3NF	BCNF	4NF
Eliminates redundancy due to FD's	Most	Yes	Yes
Eliminates redundancy due to MVD's	No	No	Yes
Preserves FD's	Yes	Maybe	Maybe
Preserves MVD's	Maybe	Maybe	Maybe

### Properties of normal forms and their decompositions

**1)First normal form:** A relation is said to be in first normal form if it contains all atomic values or single values.

Example:

Domain	Courses
Programming	C , java
Web designing	HTML , PHP

The above table consist of multiple values in single columns which can be reduced into atomic values by using first normal form as follows-

Domain	Courses
Programming	C
Programming	Java
Web designing	HTML
Web designing	PHP

**2)Second normal form:** A relation is said to be in second normal form if it is in first normal form without any partial dependencies.

→In second normal form non-prime attributes should not depend on proper subset of key attributes.

Example:

Student id	Student name	Project Id	Project name

Here (student id, project id) are key attributes and (student name, project name) are non-prime attributes. It is decomposed as-

Student id	Student name	Project id

Project id	Project name

**3)Third normal form:** A relation is said to be in third normal form , if it is already in second normal form and no transitive dependencies exists.

Transitive dependency – If  $A \rightarrow B$  and  $B \rightarrow C$  are two FDs then  $A \rightarrow C$  is called transitive dependency.

A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency  $X \rightarrow Y$

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

Student id	Student name	City	country	ZIP

It is decomposed as:

Student id	Student name	ZIP

ZIP	city	country

**4)Boyce normal form:** It is an extension of third normal form where in a functional dependency  $X \rightarrow A$  , X must be a super key.

A relation is in BCNF if in every non-trivial functional dependency  $X \rightarrow Y$ , X is a super key.

**5)fourth normal form:** A relation is said to be in fourth normal form if it is in third normal form and no multi value dependencies should exist between attributes.

Note: In some cases multi value dependencies may exist not more than one time in a given relation.

**6) fifth normal form:** fifth normal form is related to join dependencies.

→ A relation R is said to be in fifth normal form if for every join dependency JD join  $\{R_1, R_2, \dots, R_N\}$  that holds over relation R one of the following statements must be true-

1)  $R_i = R$  for some i

2) the join dependency is implied by the set of those functional dependency over relation R in which the left side is key attribute for R.

NOTE: if the relation schema is a third normal form and each of its keys consist of single attribute, we can say that it can also be in fifth normal form.

→ A join dependency JD join  $\{R_1, R_2, \dots, R_N\}$  is said to hold for a relation R if  $R_1, R_2, \dots, R_N$  this decomposition is a loss less join decomposition of R.

→ When a relation is in forth normal form and decompose further to eliminate redundancy and anomalies due to insert or update or delete operation, there should not be any loss of data or should not create a new record when the decompose tables are rejoin.

**7) Domain key normal form:** A domain key normal form keeps a constraint that every constraint on the relation is a logical sequence of definition of keys and domains.

**8) Sixth normal form:** A relation is said to be in sixth normal form such that the relation R should not contain any non-trivial join dependencies.

→ Also sixth normal form considers temporal dimensions(time) to the relational model.

**Key Points related to normal forms –**

1. BCNF is free from redundancy.
2. If a relation is in BCNF, then 3NF is also satisfied.
3. If all attributes of relation are prime attribute, then the relation is always in 3NF.
4. A relation in a Relational Database is always and at least in 1NF form.
5. Every Binary Relation ( a Relation with only 2 attributes ) is always in BCNF.
6. If a Relation has only singleton candidate keys( i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF( because no Partial functional dependency possible).
7. Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
8. There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

**problems on normal forms:****Problem 1:**

Find the highest normal form in R (A, B, C, D, E) under following functional dependencies.

ABC  $\rightarrow$  D

CD  $\rightarrow$  AE

**Solution:**

Important Points for solving above type of question.

- 1) It is always a good idea to start checking from BCNF, then 3 NF and so on.
- 2) If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, ABC  $\rightarrow$  D is in BCNF (Note that ABC is a super key), so no need to check this dependency for lower normal forms.

Candidate keys in given relation are {ABC, BCD}

BCNF: ABC  $\rightarrow$  D is in BCNF. Let us check CD  $\rightarrow$  AE, CD is not a super key so this dependency is not in BCNF. So, R is not in BCNF.

3NF: ABC  $\rightarrow$  D we don't need to check for this dependency as it already satisfied BCNF. Let us consider CD  $\rightarrow$  AE. Since E is not a prime attribute, so relation is not in 3NF.

2NF: In 2NF, we need to check for partial dependency. CD which is a proper subset of a candidate key and it determine E, which is non prime attribute. So, given relation is also not in 2NF. **So, the highest normal form is 1 NF.**

**problem 2:**

Find the highest normal form of a relation R(A,B,C,D,E) with

FD set as

{BC $\rightarrow$ D,

AC $\rightarrow$ BE,

B $\rightarrow$ E}

**Step 1:**As we can see, (AC)<sup>+</sup> = {A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.

**Step 2:** Prime attribute are those attribute which are part of candidate key {A,C} in this example and others will be non-prime {B,D,E} in this example.

**Step 3:** The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.

The relation is in 2nd normal form because BC $\rightarrow$ D is in 2nd normal form (BC is not proper subset of candidate key AC) and AC $\rightarrow$ BE is in 2nd normal form (AC is candidate key) and B $\rightarrow$ E is in 2nd normal form (B is not a proper subset of candidate key AC).

The relation is not in 3rd normal form because in BC $\rightarrow$ D (neither BC is a super key nor D is a prime attribute) and in B $\rightarrow$ E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal form, either LHS of an FD should be super key or RHS should be prime attribute.

So the highest normal form of relation will be 2nd Normal form.

**Decomposition:** It is the process of splitting original table into smaller relations such that attribute sets of two relations will be the subset of attribute set of original table.

**Rules of decomposition:**

If 'R' is a relation splitted into 'R1' and 'R2' relations, the decomposition done should satisfy following-

1) Union of two smaller subsets of attributes gives all attributes of 'R'.

$$R1(\text{attributes}) \cup R2(\text{attributes}) = R(\text{attributes})$$

2) Both relations interaction should not give null value.

$$R1(\text{attributes}) \cap R2(\text{attributes}) \neq \text{null}$$

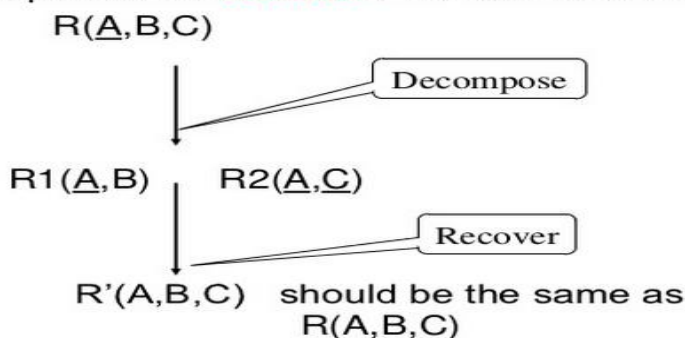
3) Both relations interaction should give key attribute.

$$R1(\text{attribute}) \cap R2(\text{attribute}) = R(\text{key attribute})$$

**Properties of decomposition:**

**Lossless decomposition:** while joining two smaller tables no data should be lost and should satisfy all the rules of decomposition. No additional data should be generated on natural join of decomposed tables.

A decomposition is *lossless* if we can recover:



----- **Must ensure R' = R** -----

**Lossless Decomposition example**

• Sometimes the same set of data is reproduced:

Name	Price	Category
Word	100	WP
Oracle	1000	DB
Access	100	DB

Name	Price
Word	100
Oracle	1000
Access	100

Name	Category
Word	WP
Oracle	DB
Access	DB

- (Word, 100) + (Word, WP) → (Word, 100, WP)
- (Oracle, 1000) + (Oracle, DB) → (Oracle, 1000, DB)
- (Access, 100) + (Access, DB) → (Access, 100, DB)



example 2 for loseless decomposition:

## Lossless Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \rightarrow B; C \rightarrow B$

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8

But, now we can't check  $A \rightarrow B$  without doing a join!

**Lossy join decomposition:** if information is lost after joining and if do not satisfy any one of the above rules of decomposition.

example 1:

## Lossy Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

$A \rightarrow B; C \rightarrow B$

A	B
1	2
4	5
7	2

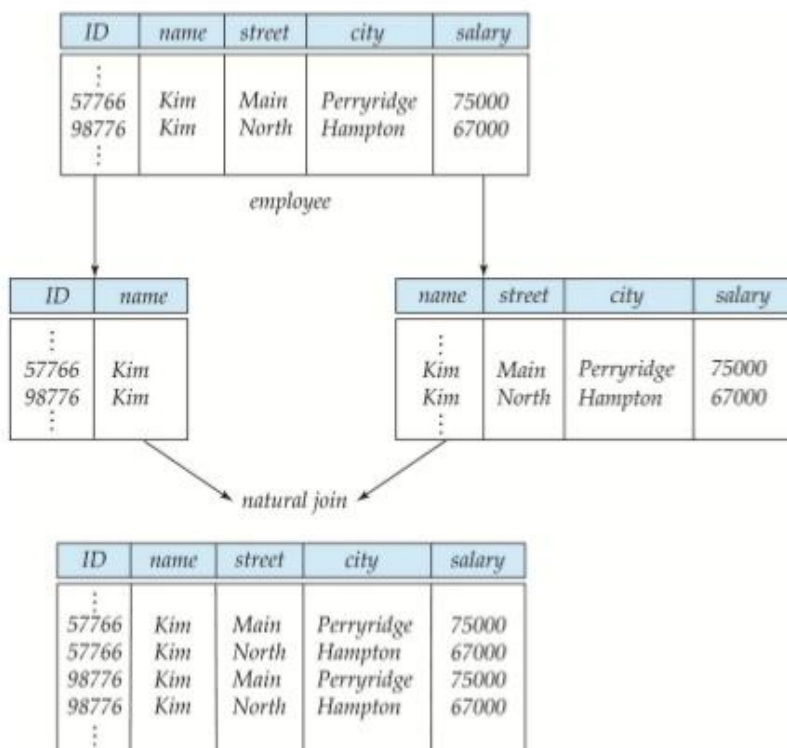
B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

example 2:

## A Lossy Decomposition



8

In above examples, on joining decomposed tables, extra tuples are generated.

so it is lossy join decomposition.

**Dependency preservation:** functional dependencies should be satisfied even after splitting relations and they should be satisfied by any of splitted tables.

### Dependency Preservation

A Decomposition  $D = \{ R_1, R_2, R_3 \dots R_n \}$  of  $R$  is dependency preserving wrt a set  $F$  of Functional dependency if

$$(F_1 \cup F_2 \cup \dots \cup F_m)^+ = F^+.$$

Consider a relation  $R$

$R \rightarrow F \{ \dots \text{with some functional dependency (FD)} \dots \}$

$R$  is decomposed or divided into  $R_1$  with FD  $\{ f_1 \}$  and  $R_2$  with  $\{ f_2 \}$ , then there can be three cases:

$f_1 \cup f_2 = F \rightarrow$  Decomposition is dependency preserving.

$f_1 \cup f_2$  is a subset of  $F \rightarrow$  Not Dependency preserving.

$f_1 \cup f_2$  is a super set of  $F \rightarrow$  This case is not possible.

example for dependency preservation:

### Dependency preservation

**Example:**

$R=(A, B, C), F=\{A \rightarrow B, B \rightarrow C\}$

Decomposition of R:  $R_1=(A, B) \quad R_2=(B, C)$

Does this decomposition preserve the given dependencies?

**Solution:**

In  $R_1$  the following dependencies hold:  $F_1=\{A \rightarrow B, A \rightarrow A, B \rightarrow B, AB \rightarrow AB\}$

In  $R_2$  the following dependencies hold:  $F_2=\{B \rightarrow B, C \rightarrow C, B \rightarrow C, BC \rightarrow BC\}$

$F' = F_1' \cup F_2' = \{A \rightarrow B, B \rightarrow C, \text{trivial dependencies}\}$

In  $F'$  all the original dependencies occur, so this decomposition preserves dependencies.

**lack of redundancy:** It is also known as repetition of information. The proper decomposition should not suffer from any data redundancy.

## UNIT 4

## TRANSACTION MANAGEMENT

**Unit 4 contents at a glance:****Transaction Management:**

- Transaction concept,
- transaction state,
- implementation of atomicity and durability,
- concurrent executions,
- Anomalies due to interleaved execution of transactions,
- serializability,
- recoverability,
- implementation of isolation

**Concurrency control and recovery system:**

- Concurrency control :
  - lock based protocols,
  - time stamp based protocols,
  - validation based protocols,
  - deadlock handling.
- Recovery system :
  - failure classification,
  - recovery and atomicity,
  - log -based recovery, shadow paging,
  - recovery with concurrent transactions,
  - ARIES algorithm

**Transaction:**

It refers to execution of any one user program in dbms.

(Or)

It can be defined as group of tasks being executed.

(Or)

It also referred to as an event that which occur on a database with read/write operation.

**PROPERTIES OF TRANSACTION(ACID PROPERTIES):**

- To ensure consistency , completeness of the database in scenario of concurrent access, system failure ,the following **ACID** properties can be enforced on to database.
  1. **Atomicity,**
  2. **Consistency,**
  3. **Isolation and**
  4. **Durability**

**Atomicity:**

- This property states that all of the instructions with in a transaction must be executed or none of them should be executed.
- This property states that all transactions execution must be atomic i.e. all actions should be carried out or none of the actions should be executed.
  - It involves following two operations.
    - Abort:** If a transaction aborts, changes made to database are not visible.
    - Commit:** If a transaction commits, changes made are visible.
 Atomicity is also known as the ‘All or nothing rule’.

**Example:**

- Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

<b>Before: X : 500</b>	<b>Y: 200</b>
Transaction T	
<b>T1</b>	<b>T2</b>
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
<b>After: X : 400</b>	<b>Y : 300</b>

- If the transaction fails after completion of **T1** but before completion of **T2**.( say, after **write(X)** but before **write(Y)**), then amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

**Consistency:**

- The database must remain in consistence state even after performing any kind of transaction ensuring correctness of the database.
- If we execute a particular transaction in isolation (or) together with other transaction in multiprogramming environment ,the transaction should give same result in any case.

- Each transaction, run by itself with no concurrent execution of other transactions, must preserve the consistency of the database. This property is called **consistency** and the DBMS assumes that it holds for each transaction. Ensuring this property of a transaction is the responsibility of the user.

**example:**

<b>Before:</b> X : 500	Y: 200
Transaction T	
<b>T1</b>	<b>T2</b>
Read (X) X: = X - 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
<b>After:</b> X : 400	Y : 300

- Referring to the example above,  
The total amount before and after the transaction must be maintained.  
Total **before T** occurs = **500 + 200 = 700**.  
Total **after T** occurs = **400 + 300 = 700**.  
Therefore, database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result T is incomplete.

**Isolation:**

- When executing multiple transactions concurrently & trying to access shared resources the system should create an order such that the only one transaction can access the shared resource at the same time & release it after completion of it's execution for other transaction.
- This property ensures that multiple transactions can occur concurrently without leading to inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed.

Note: To achieve isolation you should use locking mechanism among shared resources.

**example:**

Let **X= 500, Y = 500**.

Consider two transactions **T** and **T''**.

T	T''
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y - 50 Write	Read (X) Read (Y) Z: = X + Y Write (Z)

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result , interleaving of operations takes place due to which **T''** reads correct value of **X** but incorrect value of **Y** and sum computed by **T''**: **(X+Y = 50, 000+500=50, 500)** is thus not consistent with the sum at end of transaction:  
**T**: **(X+Y = 50, 000 + 450 = 50, 450)**.  
This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after a they have been made to the main memory.

**Durability:**

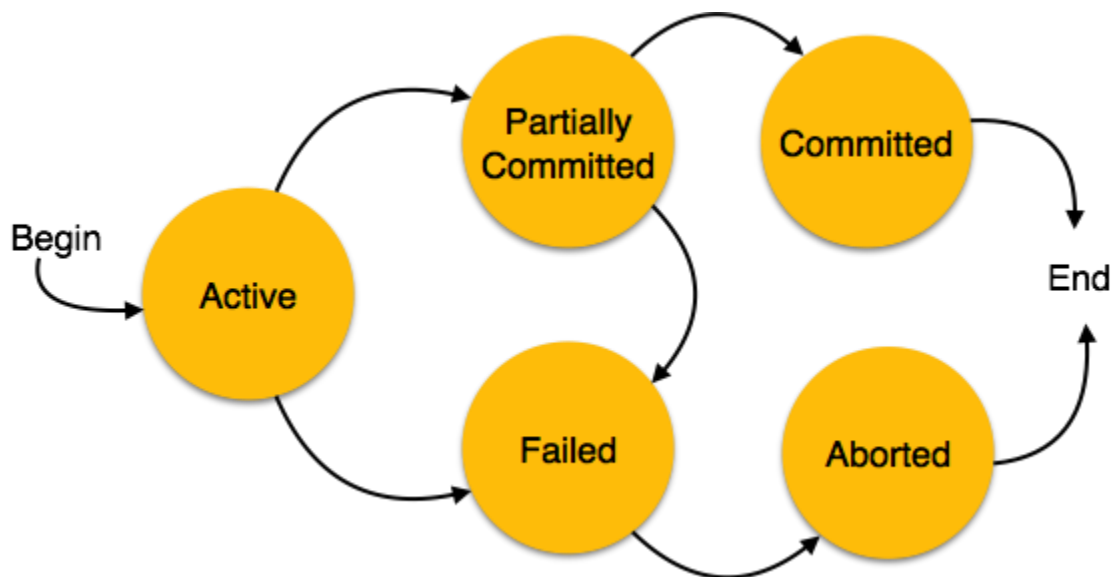
- This property states that once after the transaction is completed the changes that made should be permanent & should be recoverable even after system crash/power failure.
- This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even is system failure occurs. These updates now become permanent and are stored in a non-volatile memory.

**Transaction states:**

Every transaction undergoes several states in its execution.

A transaction can be in any one of the following states:

1. start
2. partially committed
3. committed
4. failed
5. aborted or terminate



Transaction state diagram

- **Active** - This is the first state of transaction and here the transaction is being executed. For example, updating or inserting or deleting a record is done here. But it is still not saved to the database. When we say transaction it will have set of small steps, and those steps will be executed here.
- **Partially Committed** - This is also an execution phase where last step in the transaction is executed. But data is still not saved to the database. In example of calculating total marks, final display the total marks step is executed in this state.
- **Committed** - In this state, all the transactions are permanently saved to the database. This step is the last step of a transaction, if it executes without fail.
- **Failed** - If a transaction cannot proceed to the execution state because of the failure of the system or database, then the transaction is said to be in failed state. In the total mark calculation example, if the database is not able fire a query to fetch the marks, i.e.; very first step of transaction, then the transaction will fail to execute.
- **Aborted** - If a transaction is failed to execute, then the database recovery system will make sure that the database is in its previous consistent state. If not, it brings the database to consistent state by aborting or rolling back the transaction. If the transaction fails in the middle of the transaction, all the executed transactions are rolled back to it consistent state before executing the transaction. Once the transaction is aborted it is either restarted to execute again or fully killed by the DBMS.

#### **Implementation of Durability & Atomicity:**

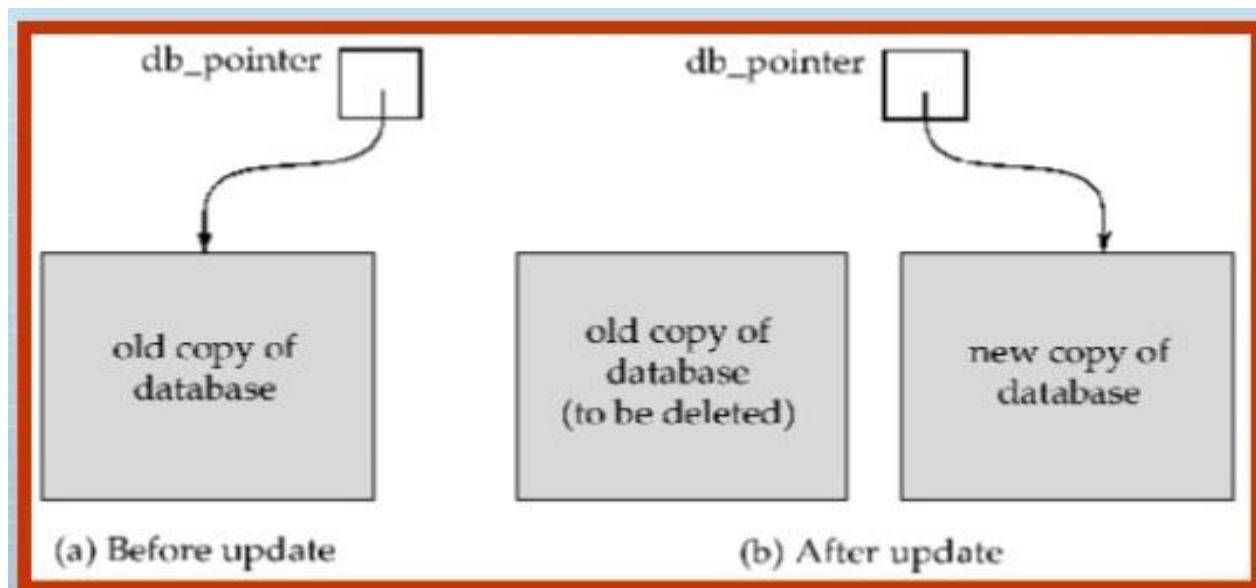
**Durability and atomicity can be ensured by using Recovery manager which is available by default in every DBMS.**

- We can **implement atomicity** by using
  1. Shadow copying technique
  2. Using recovery manager which available by default in DBMS.



### 1. Shadow copying technique:

1. Maintaining a shadow copy of original database & reflecting all changes to the database as a result of any transaction after committing the transaction.
2. The scheme also assumes that the database is simply a file on disk.
3. A pointer called db-pointer is maintained on disk; it points to the current copy of the database.
4. In the shadow-copy scheme, a transaction that wants to update the database first creates a complete copy of the database. All updates are done on the new database copy, leaving the original copy, the **shadow copy**, untouched. If at any point the transaction has to be aborted, the system merely deletes the new copy. The old copy of the database has not been affected.
5. If the transaction completes, it is committed as follows.
6. First, the operating system is asked to make sure that all pages of the new copy of the database have been written out to disk. (Unix systems use the flush command for this purpose.)
7. After the operating system has written all the pages to disk, the database system updates the pointer db-pointer to point to the new copy of the database; the new copy then becomes the current copy of the database. The old copy of the database is then deleted.



We now consider how the technique handles transaction and system failures.

First, **consider transaction failure**. If the transaction fails at any time before db-pointer is updated, the old contents of the database are not affected. We can abort the transaction by just deleting the new copy of the database. Once the transaction has been committed, all the updates that it performed are in the database pointed to by db-pointer. Thus, either all updates of the transaction are reflected, or none of the effects are reflected, regardless of transaction failure.

Now **consider the issue of system failure**. Suppose that the system fails at any time before the updated db-pointer is written to disk. Then, when the system restarts, it will read db-pointer and will thus see the original contents of the database, and none of the effects of the transaction will be visible on the database. Next, suppose that the system fails after db-pointer has been updated on disk. Before the pointer is updated, all updated pages of the new copy of the database were written to disk. Again, we assume that, once a file is written to disk, its contents will not be damaged even if there is a system failure. Therefore, when the system restarts, it will read db-pointer and will thus see the contents of the database *after* all the updates performed by the transaction.

**\*\*WE CAN IMPLEMENT DURABILITY AMONG DATA BASE USING :**

1. Recovery manager.
  2. Logs
- Partial transaction should be avoided for ensuring atomicity and durability.

**LOGS:**

- Logs keep track of actions carried out by transactions which can be used for the recovery of database in case of failure.
- Logs files should be stored always on stable storage devices.
- When a transaction begins its execution it is recorded in the log as follows

<Tn, start>

- When a transaction performs an operation it is recorded in log as follows

<Tn, X, V1, V2>

- When a transaction finishes its execution, it is recorded as  
    <Tn,commit>

**CONCURRENT EXECUTION:**

Executing a set of transactions simultaneously in a pre-emptive and time-shared method.

In DBMS concurrent execution of transaction can be implemented with interleaved execution.

**TRANSACTION SCHEDULES:**

**Schedule:**

- It refers to the list of actions to be executed by transaction.
- A **schedule** is a process of grouping the transactions into one and executing them in a predefined order.
- Schedule of actions can be classified into 2 types.
  1. Serializable schedule/serial schedule.
  2. Concurrent schedule.

**1. Serial schedule:**

In the serial schedule the transactions are allowed to execute one after the other ensuring correctness of data.

A schedule is called serial **schedule**, if the transactions in the schedule are defined to execute one after the other.

**2. Concurrent schedule:**

Concurrent schedule allows the transaction to be executed in interleaved manner of execution.

**Complete schedule:**

It is a schedule of transactions where each transaction is committed before terminating. The example is shown below where transactions T1 and T2 terminate after committing the transactions.

Example:

T1	T2
A=1000	
Read(A)	
A=A+100	
Write(A)	Read(A)
	B=A-100
	Write(B)
	Commit
Read(B)	
Write(B)	
Commit	

**SERIALIZABILITY:**

A transaction is said to be **Serializable** if it is equivalent to serial schedule.

Serializability aspects are:

1. Conflict serializability.
2. View serializability.

**1. Conflict serializability:**

A schedule is **conflict serializable** if it is conflict equivalent to some serial schedule.

**Conflict Equivalent:** Two schedules are said to be conflict equivalent when one can be transformed to another by swapping non-conflicting operations.

**Conflict Serializable:** A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

**Conflicting operations:** Two operations are said to be conflicting if all below conditions are satisfied:

- They belong to different transaction
- They operation on same data item
- At Least one of them is a write operation

it refers to two instructions of two different transactions may want to access same data to perform read/write operation.

**Rules for conflict serializability:**

- If two different transactions are both for read operation, then there is no conflict and can allowed to execute any order.
- If one instruction performing read operation and other instruction performing write operation there will be conflict hence instruction ordering is important.
- If both transactions performing write operation then there will be in conflict so ordering the transaction can be done.

## 2. View serializability:

This is another type of serializability that can be derived by creating another schedule out of an existing Schedule.

A schedule is **view serializable** if it is view equivalent to some serial schedule. Every conflict serializable schedule is view serializable, although the converse is not true.

Two schedules  $S_1$  and  $S_2$  over the same set of transactions --any transaction that appears in either  $S_1$  or  $S_2$  must also appear in the other are **view equivalent** under these conditions:

1. If  $T_i$  reads the initial value of object  $A$  in  $S_1$ , it must also read the initial value of  $A$  in  $S_2$ .
2. If  $T_i$  reads a value of  $A$  written by  $T_j$  in  $S_1$ , it must also read the value of  $A$  written by  $T_j$  in  $S_2$ .
3. For each data object  $A$ , the transaction (if any) that performs the final write on  $A$  in  $S_1$  must also perform the final write on  $A$  in  $S_2$ .

- The above two schedules are view serializable or view equivalence, if the transactions in both schedules performs the actions in similar manner.
- The above two schedules satisfy result view equivalence if the two schedule produces the same Result after execution.

Ex:

$s_1: R_1(A), W_1(A), R_2(A), W_2(A), R_1(B), W_1(B), R_2(B), W_2(B)$

**Anomalies due to interleave execution of transaction:**

Due to interleaved execution of transaction the following anomalies can occur

1. reading uncommitted values(WR conflicts)
2. un repeatable reading data operation(RW conflicts)
3. Overwriting uncommitted data(WW )

**1. reading uncommitted values(WR conflicts):**

- If you try to the read the value which is not written on to the data base(not committed) will leads to write-read conflict which is called **dirty read operation**.

<i>T1</i>	<i>T2</i>
<i>R(A)</i>	
<i>W(A)</i>	
	<i>R(A)</i>
	<i>W(A)</i>
	<i>R(B)</i>
	<i>W(B)</i>
	Commit
<i>R(B)</i>	
<i>W(B)</i>	
Commit	

**Figure 18.2** Reading Uncommitted Data

In above example, T1 write operation on data item A is not committed but it is being read by T2. So reading an uncommitted data will leads to inconsistency in database which is called dirty read operation.

**2. un repeatable reading data operation(RW conflicts):**

*Reading the same object twice before committing the transaction might yield an inconsistency*

–Read-then-Write (RW) Conflicts (Write-After-Read)

*Unrepeatable problem means we get different values in different reads.* For example in S1 say T2 read initially x=5 then T1 updated x=1 so now T2 will read x=1 here T2 has read two different values during consecutive reads This shouldn't have been allowed as T1 has not committed

### 3. Overwriting uncommitted data(WW conflicts)

WW conflicts if one transaction could over write the value of an object A which has been already modified by other transaction while first transaction still in progress .this kind of conflict refer to **blind write conflict**.

#### Recoverability:

It refers to the process of undoing the changes made to the database in case of any transaction failure due to system crash or any other reason.

#### Recoverability Schedule:

Based on whether recovery of failure transaction schedules are classified as

1. Irrecoverable schedules.
2. Recoverable schedules with cascade rollback.
3. Cascade less recoverability.

#### 1. Irrecoverable schedules: schedules which can't be recovered

- If transaction T2 read the value updated by Transaction T1 followed by write operation commit then this schedule is called Irrecoverable Schedule. If transaction1 failed before committing

Example:

T1	T1's buffer space	T2	T2's buffer space	database
R(A)	A=5000			A=5000
A=A-100	A=4000			A=5000
W(A)	A-4000			A=5000
		<b>R(A)</b>	A=4000	A=4000
		<b>A=A+500</b>	A=4500	A=4000
		<b>W(A)</b>	A=4500	A=4000
		<b>Commit;</b>		
Failure point				A=4000
Commit				A=4500

**2. Recoverable schedule with cascade rollback: schedules which can be recoverable**

Example:

T1	T1's buffer space	T2	T2's buffer space	database
R(A)	A=5000			A=5000
A=A-100	A=4000			A=5000
W(A)	A=4000			A=5000
		<b>R(A)</b>	A=4000	A=4000
		<b>A=A+500</b>	A=4500	A=4000
		<b>W(A)</b>	A=4500	A=4000
Failure point				A=4000
Commit				A=4500

- IF transaction T2 reading a value updated by T1 & commit of T2 is delay till the commit of T1, it is called recoverable schedule with cascade roll back.

**3. Cascade less recoverability:**

It refers to if T2 read value updated by T1 only after T1 is committed.

Example:

T1	T1's buffer space	T2	T2's buffer space	database
R(A)	A=5000			A=5000
A=A-100	A=4000			A=5000
W(A)	A=4000			A=5000
commit		<b>R(A)</b>	A=4000	A=4000
		<b>A=A+500</b>	A=4500	A=4000
		<b>W(A)</b>	A=4500	A=4000
		<b>Commit;</b>		
Failure point				A=4000
				A=4500



**Implementation of Isolation:**

- When more than one instruction of several transaction are being executed concurrently by using some sharable resources, the execution of instruction of one transaction should not interrupt the execution of instruction of another transaction.
- 1. **Access to sharable resources should be order by using some locking mechanism:**  
Where one transaction locks the sharable resource before starting its execution & release the lock to other transaction after completion of its execution.
- 2. **Locking protocols:**  
Locking mechanism can be implemented by using locking protocols which defined set of standard rule based on which transaction access, sharable resources.

**Transaction control commands supported with SQL:**

1. Commit.
2. Save point.
3. Roll back.

explain about usage of above 3 commands with syntaxes.

**Precedence graph in serializability:**

Precedence graph or serializability graph is used commonly to test conflict serializability of a schedule.

- It is a directed graph which consist of nodes  $G(V,E)$  where nodes(v) represents set of transaction & E represents set of edges  $\{E_1,E_2,\dots,E_n\}$ .
- The graph contains one node for each transaction  $T_i$ . Each edge  $E_i$  is of the form  $T_j \rightarrow T_k$  Where  $T_j$  is starting node of edge j &  $T_k$  is ending node of edge k.
- An edge is constructed between nodes if one of the operation in transaction  $T_j$  appear in the schedule before some conflicting operation in transaction  $T_k$ .

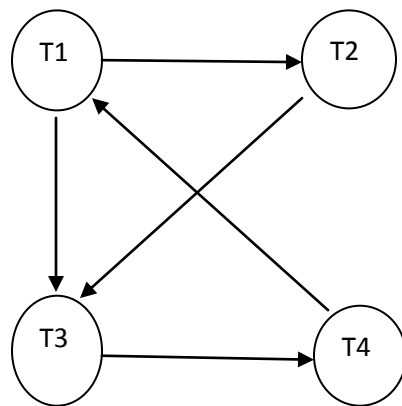
**Algorithm:**

1. Create a node  $T_n$  in the graph for each participating transaction in the schedule.
2. Draw edges from one transaction to another transaction when satisfy anyone of the following condition.
  - Condition 1:
    - If  $T_1$  execute write operation i.e.  $write(x)$  followed by  $T_2$  execute read operation i.e.  $read(x)$ .
  - Condition 2:
    - When  $T_1$  executes  $read(x)$  followed by  $T_2$  execute  $write(x)$ .
  - Condition 3:
    - When  $T_1$  execute  $write(x)$  followed by  $T_2$  execute  $write(x)$ .
3. The given schedule is serializable if there are no cycles in the precedence graph.

Example for precedence graph:

draw precedence graph for below transaction schedule.

T1	T2	T3	T4
Read(x)			
	Read(x)		
Write(x)			
		Read(y)	
	Read(y)		
	Write(x)		
		Read(w)	
		Write(y)	
			Read(w)
			Read(z)
			Write(w)
Read(z)			
Write(z)			



As precedence graph is having cycles or closed loops, the given schedule is not serializable.

**Example of conflict serializability:**

S2:R1(X), R2(X), R2(Y), W2(Y), R1(Y), W1(X)

**Sol:**

S21:R2(X), R1(X),R2(Y),W2(Y),R1(Y),W1(Y)

S22:R2(X),R2(Y),R1(X),W2(Y),R1(Y),W1(Y)

S23:R2(X),R2(Y),W2(Y),R1(X),R1(Y),W1(Y)

The schedule S2 derives 3 more schedules (s21,s22,s23) which is called **conflict equivalence**

**Concurrency Control:**

In case of concurrent instruction executions to preserve atomicity, isolation and serializability, we use 'lock-based' protocol like .

**Types of Locks:**

1. Binary locks
2. Shared /exclusive locks

- **Binary Locks** – A lock on a data item can be in two states; it is either locked or unlocked.
- **Shared(S)/exclusive(X)** – This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

**Lock Compatibility Matrix –**

- Lock Compatibility Matrix controls whether multiple transactions can acquire locks on the same resource at the same time.

	Shared	Exclusive
Shared	True	False
Exclusive	False	False

- If a resource is already locked by another transaction, then a new lock request can be granted only if the mode of the requested lock is compatible with the mode of the existing lock.
- Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive lock on item, no other transaction may hold any lock on the item.
- compatible locks held by other transactions have been released. Then the lock is granted.

**Lock Granularity :**

A database is basically represented as a collection of named data items. The size of the data item chosen as the unit of protection by a concurrency control program is called GRANULARITY.

Locking can take place at the following level :

- Database level.
- Table level.
- Page level.
- Row (Tuple) level.
- Attributes (fields) level.

**i. Database level Locking :**

At database level locking, the entire database is locked. Thus, it prevents the use of any tables in the database by transaction T2 while transaction T1 is being executed. Database level of locking is suitable for batch processes. Being very slow, it is unsuitable for on-line multi-user DBMSs.

**ii. Table level Locking :**

At table level locking, the entire table is locked. Thus, it prevents the access to any row (tuple) by transaction T2 while transaction T1 is using the table. If a transaction requires access to several tables, each table may be locked. However, two transactions can access the same database as long as they access different tables. Table level locking is less restrictive than database level. Table level locks are not suitable for multi-user DBMS

**iii. Page level Locking :**

At page level locking, the entire disk-page (or disk-block) is locked. A page has a fixed size such as 4 K, 8 K, 16 K, 32 K and so on. A table can span several pages, and a page can contain several rows (tuples) of one or more tables. Page level of locking is most suitable for multi-user DBMSs.

**iv. Row (Tuple) level Locking :**

At row level locking, particular row (or tuple) is locked. A lock exists for each row in each table of the database. The DBMS allows concurrent transactions to access different rows of the same table, even if the rows are located on the same page. The row level lock is much less restrictive than database level, table level, or page level locks. The row level locking improves the availability of data. However, the management of row level locking requires high overhead cost.

**v. Attributes (fields) level Locking :**

At attribute level locking, particular attribute (or field) is locked. Attribute level locking allows concurrent transactions to access the same row, as long as they require the use of different attributes within the row. The attribute level lock yields the most flexible multi-user data access. It requires a high level of computer overhead.

**Locking protocols:**

1. Simple lock based protocol
2. Conservative (or) pre-claim locking protocol.
3. 2-phase locking protocol
4. Strict 2 phase locking protocol
5. Rigorous 2 phase locking protocol

**Simple lock based protocol:**

Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the 'write' operation.

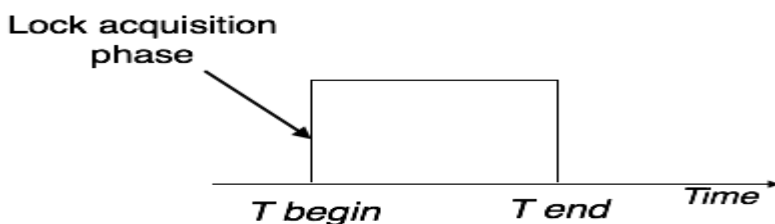
problems with simple locking are:

1. deadlocks
2. starvation

**Conservative (or) pre-claim locking protocol:**

Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand.

If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.



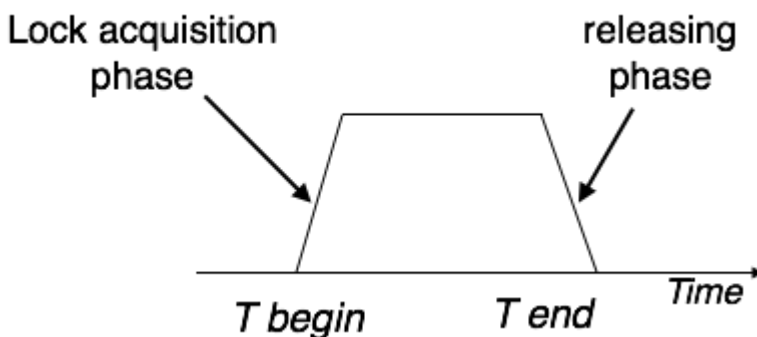
**2-phase locking protocol:**

This locking protocol divides the execution phase of a transaction into three parts.

- In the first part, when the transaction starts executing, it seeks permission for the locks it requires.
- The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts.
- In third phase, the transaction cannot demand any new locks; it only releases the acquired locks.

**This protocol can be divided into two phases,**

- 1. In Growing Phase,** a transaction obtains locks, but may not release any lock.
- 2. In Shrinking Phase,** a transaction may release locks, but may not obtain any lock.



Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.

To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

***Types of Two – Phase Locking Protocol***

**Following are the types of two – phase locking protocol:**

1. Strict Two – Phase Locking Protocol
2. Rigorous Two – Phase Locking Protocol
3. Conservative Two – Phase Locking Protocol

**Strict Two-Phase Locking:**

1. If a transaction want to read any value it can refers to a shared lock
2. If a transaction to write any particular value it can refers to an exclusive locks
3. A shared lock acquire by multiple transaction at same time.
4. An exclusive lock can be requested by only one transaction at a time on any data item.
5. Strict Two-Phase Locking Protocol avoids cascaded rollbacks.
6. It ensures that if data is being modified by one transaction, then other transaction cannot read it until first transaction commits.

phases in strict 2 phase locking:

**phase 1:** The first phase of Strict-2PL is same as 2PL i.e. when the transaction starts executing, it seeks permission for the locks it requires.

**phase 2:**After acquiring all the locks in the first phase, the transaction continues to execute normally.

**phase 3:** But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.

**Note:** It releases only all exclusive locks but not shared locks after a transaction is committed .

This protocol is not free from deadlocks

**Rigorous Two-Phase Locking**

- Rigorous Two – Phase Locking Protocol avoids cascading rollbacks.
- This protocol requires that all the share and exclusive locks to be held until the transaction commits.
- it releases all the locks including shared and exclusive locks after committing the transactions.
- It considers the order of commit among transaction executions.

**Conservative Two-Phase Locking Protocol**

- Conservative Two – Phase Locking Protocol is also called as Static Two – Phase Locking Protocol.
- This protocol is almost free from deadlocks as all required items are listed in advanced.
- It requires locking of all data items to access before the transaction starts.



**UPGRADING AND DOWNGRADING of Locks:**

- If a transaction is holding an exclusive lock over an object .It can simply **downgrade** from exclusive lock to shared lock after completion of its updation
- Similarly a shared lock can be **upgraded** to exclusive lock on particular data item. when there is no other transaction is holding exclusive lock on same data item
- Strict 2 phase locking protocol can be executed serial/concurrent execution of transaction
- examples for serial and concurrent execution are shown below:

T1	T2
S(A)	
R(A)	
X(A)	
W(A)	
COMMIT	
	X(A)
	W(A)
	COMMIT
Serial	

T1	T2
S(A)	
R(A)	
	X(A)
	W(A)
	commit
X(A)	
W(A)	
commit	
Concurrent	

**IMPLEMENTING LOCKS:**

- Every DBMS maintains a **lock manager** which maintain two tables called **lock table** and **transaction table**
- Lock table consist of information regarding locks on data item holding:
  1. No. of transaction holding lock
  2. Nature of lock(shared or exclusive)
  3. Pointer to the no. of locks requested in queue in given object.
- Transaction table:  
Transaction table contain list of transactions and their corresponding locks assigned.

**TIME STAMP BASED PROTOCOLS:**

- The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.
- It starts working as soon as a transaction is created.
- Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction.
- every data item is given the latest read and write-timestamp.
- This lets the system know when the last 'read and write' operation was performed on the data item.
- Each transaction is issued a timestamp when it enters into the system.
- Every read and write operations will be marked with a time stamp of their occurrence.
- Timestamp Based Protocol helps DBMS to identify the transactions.
- Time stamp is a unique identifier.
- Timestamp protocol determines the serializability order.
- It is most commonly used concurrency protocol.
- It uses either system time or logical counter as a timestamp.

**Timestamp Ordering Protocol**

- The TO Protocol ensures serializability among transactions in their conflicting read and write operations.
- The transaction of timestamp (T) is denoted as TS(T).
- Data item (X) of read timestamp is denoted by R-timestamp(X).
- Data item (X) of write timestamp is denoted by W-timestamp(X).

**The below assumptions in Time stamp based ordering protocol are based on THOMAS WRITE RULE.**

***If a transaction  $T_i$  issues a read(X) operation***

- If  $TS(T_i) < \text{Write-timestamp}(x)$  , then Operation rejected  
If  $TS(T_i) \geq \text{Write-timestamp}(x)$ , then Operation executed  
All data items time stamps updated
- ***If a transaction  $T_i$  issues write(X) operation***  
If  $TS(T_i) < \text{Read-Timestamp}(x)$ , then operation rejected  
If  $TS(T_i) < \text{Write-timestamp}(x)$ , then operation rejected &  $T_i$  rolled back  
Otherwise operation executed

**Thomas' Write Rule**

This rule states if  $TS(T_i) < W\text{-timestamp}(X)$ , then the operation is rejected and  $T_i$  is rolled back.

Time-stamp ordering rules can be modified to make the schedule view serializable.

Instead of making  $T_i$  rolled back, the 'write' operation itself is ignored.

Following are the three basic variants of timestamp-based methods of concurrency control:

- Total timestamp ordering
- Partial timestamp ordering
- Multiversion timestamp ordering

***Total timestamp ordering :***

The total timestamp ordering algorithm depends on maintaining access to granules in timestamp order by aborting one of the transactions involved in any conflicting access.

***Partial timestamp ordering :***

In a partial timestamp ordering, only non-permutable actions are ordered to improve upon the total timestamp ordering. In this case, both Read and Write granule timestamps are stored. The algorithm allows the granule to be read by any transaction younger than the last transaction that updated the granule. A transaction is aborted if it tries to update a granule that has previously been accessed by a younger transaction.

***Multiversion Timestamp ordering :***

The multiversion timestamp ordering algorithm stores several versions of an updated granule, allowing transactions to see a consistent set of versions for all granules it accesses. So, it reduces the conflicts that result in transaction restarts to those where there is a Write-Write conflict.

**VALIDATION BASED PROTOCOLS:**

These are also called as optimistic concurrency control method.

An optimistic concurrency control method is also known as validation or certification methods. No checking is done while the transaction is executing. The optimistic method does not require locking or timestamping techniques. Instead, a transaction is executed without restrictions until it is committed.

In validation based protocols every transaction is executed on 3 bases

1. read phase
2. validation phase
3. execute or write phase

**1. Read phase:**

In this phase transaction is executed and all the result will be stored in temporary variables local to transactions.

**2. validation phase:**

In this phase the transaction operations are validated without violating the serializability.

**3. write phase:**

In this phase when a transaction is validated successfully all the values of temporary variables is updated in the actual data base.

**Validation phase:**

A transaction is validated based on following time stamp

**1. start(ti):**

The time at which the transaction  $t_i$  started its execution.

**2. validation(ti):**

The time at which  $t_i$  is valid.

**3. finish(ti):**

The time at which  $t_i$  finish its write operation on the actual data base its execution.

Among two transactions  $t_i$  &  $t_j$ , the transactions  $t_i$  is validated. If it satisfy one of the two conditions.

If for all  $t_i$  with  $ts(t_i) < ts(t_j)$

1.  $finish(t_i) < start(t_j)$
2.  $Start(t_j) < finish(t_i) < validation(t_j)$

- The below example shows the interleaved execution of 3 phases of 2 transactions in which transaction t14 is validated.

T14	T15
Read(B)	
	Read(B)
	B=B-50
	Read(A)
	A=A+50
Read(A)	
Validate	
Display(A+B)	
	Validate
	Write(B)
	Write(A)

**Advantages of Optimistic Methods for Concurrency Control :**

- This technique is very efficient when conflicts are rare. The occasional conflicts result in the transaction roll back.
- The rollback involves only the local copy of data, the database is not involved and thus there will not be any cascading rollbacks.

**Problems of Optimistic Methods for Concurrency Control :**

- Conflicts are expensive to deal with, since the conflicting transaction must be rolled back.
- Longer transactions are more likely to have conflicts and may be repeatedly rolled back because of conflicts with short transactions.

**DEAD LOCKS:**

Consider two transaction  $t_1$  and  $t_2$ . If  $t_1$  holds lock on data item  $x$  and  $t_2$  holds lock on data item  $y$  now  $t_1$  requests lock over  $y$  &  $t_2$  requests lock over  $x$  then **deadlock situation** occurs when none of the transactions are ready to release locks on  $x, y$ .

- The following two techniques can be used for deadlock handling (prevention):

1. wait-die
2. wait-wound

**1. wait-die:**

- In wait-die technique the older transaction waited in queue & younger will die.
  - The older transaction waits for the younger if the younger has accessed the granule first.
  - The younger transaction is aborted (dies) and restarted if it tries to access a granule after an older concurrent transaction.
- The wait-die is based on time stamp of the transaction request for conflicting resources.

1)  $Ts(t_1) < ts(t_2)$ :  $t_1$  will wait in a queue &  $t_2$  will die/abort.

2)  $ts(t_1) > ts(t_2)$ :  $t_2$  will be waiting in queue &  $t_1$  will abort/die

**For example:**

Suppose that transaction  $T_{22}, T_{23}, T_{24}$  have time-stamps 5, 10 and 15 respectively. If  $T_{22}$  requests a data item held by  $T_{23}$  then  $T_{22}$  will wait. If  $T_{24}$  requests a data item held by  $T_{23}$ , then  $T_{24}$  will be rolled back.

**2. wait wound technique:**

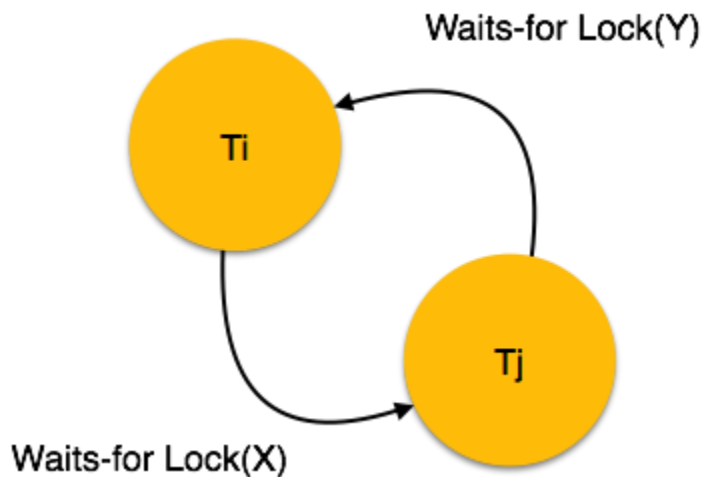
- It is based on time stamp of transaction request
  - It is a preemptive technique for deadlock prevention. It is a counterpart to the wait-die scheme. When Transaction  $T_i$  requests a data item currently held by  $T_j$ ,  $T_i$  is allowed to wait only if it has a timestamp larger than that of  $T_j$ , otherwise  $T_j$  is rolled back ( $T_j$  is wounded by  $T_i$ )

**For example:**

- Suppose that Transactions  $T_{22}, T_{23}, T_{24}$  have time-stamps 5, 10 and 15 respectively. If  $T_{22}$  requests a data item held by  $T_{23}$ , then data item will be preempted from  $T_{23}$  and  $T_{23}$  will be rolled back. If  $T_{24}$  requests a data item held by  $T_{23}$ , then  $T_{24}$  will wait.
- Here the younger transactions are made to wait in queue & older transaction going to abort.
  - 1)  $Ts(t_1) < ts(t_2)$ :  $t_2$  will be in waiting state &  $t_1$  in abort.
  - 2)  $Ts(t_1) > ts(t_2)$ :  $t_1$  will be in waiting &  $t_2$  in abort.

**DEAD LOCK AVOIDANCE:****Wait for graph:**

- We use this technique for dead lock avoidance.
- This is a simple method available to track if any deadlock situation may arise.
- For each transaction entering into the system, a node is created.
- When a transaction  $T_i$  requests for a lock on an item, say X, which is held by some other transaction  $T_j$ , a directed edge is created from  $T_i$  to  $T_j$ . If  $T_j$  releases item X, the edge between them is dropped and  $T_i$  locks the data item.
- The system maintains this wait-for graph for every transaction waiting for some data items held by others. The system keeps checking if there's any cycle in the graph.



Here, we can use any of the two following approaches –

- First, do not allow any request for an item, which is already locked by another transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.
- The second option is to roll back one of the transactions. It is not always feasible to roll back the younger transaction, as it may be important than the older one. With the help of some relative algorithm, a transaction is chosen, which is to be aborted. This transaction is known as the **victim** and the process is known as **victim selection**.

**CRASH RECOVERY:**

- In the case of DBMS, durability is a key property along with atomicity.
- **Failure Classification in DBMS:**
  1. when a transaction is failed
    - 1) Logical error
    - 2) System error
  2. system crash
  3. disk failure
  4. storage structure
    - 1) volatile
    - 2) non volatile

**Transaction failure**

A transaction has to abort when it fails to execute or when it reaches a point from where it can't go any further. This is called transaction failure where only a few transactions or processes are hurt.

**Transaction failure :**

- ✎ Logical errors: transaction cannot complete due to some internal error condition
- ✎ System errors: the database system must terminate an active transaction due to an error condition (e.g., deadlock)
- ✎ System crash: a power failure or other hardware or software failure causes the system to crash. It is assumed that non-volatile storage contents are not corrupted.
- ✎ Disk failure: a head crash or similar failure destroys all or part of disk storage

**Storage Structure:**

- **Volatile storage:**
  - ✎ does not survive system crashes
  - ✎ examples: main memory, cache memory
- **Nonvolatile storage:**
  - ✎ survives system crashes
  - ✎ examples: disk, tape
- **Stable storage:**
  - ✎ a mythical form of storage that survives all failures
  - ✎ approximated by maintaining multiple copies on distinct nonvolatile media



## Comparison between Volatile & Non-volatile Memory

<b>Volatile Memory</b>	<b>Non-volatile Memory</b>
✓ Information stored is lost if power turns off.	✓ Information stored is does not lost if power turns off.
✓ Types – All RAMs, SRAM, DRAM	✓ Types - All ROMs, EPROM, EEPROM
✓ Used for temporary storage	✓ Used for permanent storage
✓ Uses mainly Solid state devices	✓ Uses magnetic, optical or sold state devices
✓ Fast operation	✓ Slow operation

### **Recovery of data:**

When a database is recovered after a failure it should ensure the atomicity property & following should be done after a crash.

- 1) we should check the status of all transactions whether they are executed completely or partially
- 2) Check for the transaction which are in the middle of execution & should take care of atomicity property with transaction.
- 3) We should check whether there are any transactions which can be completed after recovery.
- 4) If such transactions are there we should be rollback to previous commit point that allowed for execution.
- 5) The recovery of database can be done in 2 ways:
  1. By using logs
  2. by using shadow paging technique.

**Log based recovery:**

- Logs keep track of actions carried out by transactions which can be used for the recovery of database in case of failure.
- Logs files should be stored always on stable storage devices.
- When a transaction begins its execution it is recorded in the log as follows

<Tn, start>

- When a transaction performs an operation it is recorded in log as follows

<Tn, X, V1, V2>

- When a transaction finishes its execution, it is recorded as

<Tn,commit>

- By using logs with in DBMS the updation to the database can occur in 2 ways
  - 1) Differed database updation(database is updated only after committing the transaction)
  - 2) Immediate database updation(updating database will be done immediately after execution of instructions without waiting for commit).
  - While recovering the data about transaction by using log files each transaction will be listed in one of the below list.
    1. re-do list
    2. undo list

No.	Deferred update recovery	Immediate update recovery
1	It is a protocol that the updates are written to the database only when transactions are committed.	It is a protocol that the updates are applied to the database as they occur without waiting for commit statement.
2	In case of transaction failure, just redo the operations of committed transactions.	In case of transaction failure, redo the committed transactions and at the same time undo the operations of uncommitted transaction.
3	For a write operation just update the log file, do not actually write the record in database.	For a write operation, update the log file and write the update to database buffer.
4	For a commit operation, update the log and perform actual update to the database.	For a commit operation just update the log file.
5	If the transaction aborts, just update the log record, do not perform any update in database.	If a transaction aborts, undo all the operations in the transaction using the log file as it contains the previous values.

## Shadow Paging:

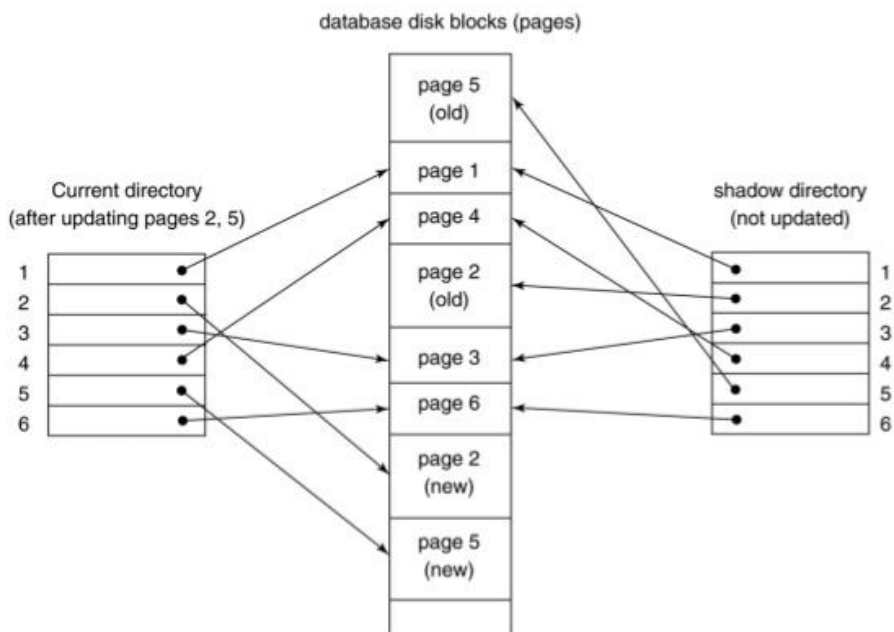
### Concept of Shadow Paging Technique

- Shadow paging is an alternative to transaction-log based recovery techniques.
- Here, the database considered as made up of fixed size disk blocks, called pages. These pages mapped to physical storage using a table, called page table.
- The page table indexed by a page number of the database. The information about physical pages, in which database pages are stored, is kept in this page table.
- This technique is similar to paging technique used by Operating Systems to allocate memory, particularly to manage virtual memory.
- The following figure depicts the concept of shadow paging.

### Execution of Transaction

- During the execution of the transaction, two-page tables maintained.
  1. **Current Page Table:** Used to access data items during transaction execution.
  2. **Shadow Page Table:** Original page table, and not get modified during transaction execution.
- Whenever any page is about to written for the first time
  1. A copy of this page made into a free page,
  2. The current page table made to point to the copy,
  3. The update made in this copy.

**FIGURE 19.5** An example of shadow paging.



# Shadow Paging

- In this technique, the database is considered to be made up of fixed-size disk blocks or pages for recovery purposes.
- Maintains two tables during the lifetime of a transaction-current page table and shadow page table.
- Store the shadow page table in nonvolatile storage, to recover the state of the database prior to transaction execution
- This is a technique for providing atomicity and durability.

When a transaction begins executing





**ARIES Recovery Algorithm:**

- A steal, no-force approach
  - Steal: if a frame is dirty and chosen for replacement, the page it contains is written to disk even if the modifying transaction is still active.
  - No-force: Pages in the buffer pool that are modified by a transaction are not forced to disk when the transaction commits.

**Algorithms for Recovery and Isolation Exploiting Semantics, or ARIES** is a recovery algorithm designed to work with a no-force, steal database approach.

The ARIES recovery procedure consists of three main steps:

**1. Analysis**

The analysis step identifies the dirty (updated) pages in the buffer, and the set of transactions active at the time of the crash. The appropriate point in the log where the REDO operation should start is also determined

**2. REDO**

The REDO phase actually reapplies updates from the log to the database. Generally, the REDO operation is applied to only committed transactions. However, in ARIES, this is not the case. Certain information in the ARIES log will provide the start point for REDO, from which REDO operations are applied until the end of the log is reached. In addition, information stored by ARIES and in the data pages will allow ARIES to determine whether the operation to be redone has actually been applied to the database and hence need not be reapplied. Thus only the necessary REDO operations are applied during recovery.

**3. UNDO**

During the UNDO phase, the log is scanned backwards and the operations of transactions that were active at the time of the crash are undone in reverse order. The information needed for ARIES to accomplish its recovery procedure includes the log, the Transaction Table, and the Dirty Page Table. In addition, check pointing is used. These two tables are maintained by the transaction manager and written to the log during check pointing.

**Data structures used in ARIES algorithm:**

1. page table
2. dirty page table
3. pageLSN
4. RedoLSN
5. Transaction Table
6. Checkpoint Log

\*\* LSN stands for Log Sequence Number

For efficient recovery, we need Transaction table and Dirty Page table .

The above 2 tables are maintained by transaction manager

The **Transaction Table** contains an entry for *each active transaction*, with information such as the transaction ID, transaction status, and the LSN of the most recent log record for the transaction.

Transaction ID	Transaction Status	LSN of recent log

The **Dirty Page Table** contains an entry for each dirty page in the buffer, which includes the page ID and the LSN corresponding to the earliest update to that page.

PageID	LSN of earliest update to page

**Checkpointing** in ARIES consists of the following:

1. writing a begin\_checkpoint record to the log,
2. writing an end\_checkpoint record to the log, and
3. writing *the LSN of* the begin\_checkpoint record to a special file.

This Checkpoint log file is accessed during recovery to locate the last checkpoint information.

**After a crash, the ARIES recovery manager takes over.**

Information from the last checkpoint is first accessed through the special file. The **analysis phase** starts at the `begin_checkpoint` record and proceeds to the end of the log. When the `end_checkpoint` record is encountered, the Transaction Table and Dirty Page Table are accessed (recall that these tables were written in the log during checkpointing). During analysis, the log records being analyzed may cause modifications to these two tables. For instance, if an end log record was encountered for a transaction  $T$  in the Transaction Table, then the entry for  $T$  is deleted from that table. If some other type of log record is encountered for a transaction  $T$ , then an entry for  $T$  is inserted into the Transaction Table, if not already present, and the last LSN field is modified. If the log record corresponds to a change for page  $P$ , then an entry would be made for page  $P$  (if not present in the table) and the associated LSN field would be modified. When the analysis phase is complete, the necessary information for REDO and UNDO has been compiled in the tables.

The REDO **phase** follows next.

ARIES starts redoing at a point in the log where it knows (for sure) that previous changes to dirty pages *have already been applied to the database on disk*. It can determine this by finding the smallest LSN,  $M$ , of all the dirty pages in the Dirty Page Table, which indicates the log position where ARIES needs to start the REDO phase. Any changes corresponding to an LSN  $< M$ , for redoable transactions, must have already been propagated to disk or already been overwritten in the buffer; otherwise, those dirty pages with that LSN would be in the buffer (and the Dirty Page Table). So, REDO starts at the log record with LSN =  $M$  and scans forward to the end of the log. For each change recorded in the log, the REDO algorithm would verify whether or not the change has to be reapplied. For example, if a change recorded in the log pertains to page  $P$  that is not in the Dirty Page Table, then this change is already on disk and does not need to be reapplied. Or, if a change recorded in the log (with LSN =  $N$ , say) pertains to page  $P$  and the Dirty Page Table contains an entry for  $P$  with LSN greater than  $N$ , then the change is already present. If neither of these two conditions hold, page  $P$  is read from disk and



the LSN stored on that page,  $LSN(P)$ , is compared with  $N$ . If  $N < LSN(P)$ , then the change has been applied and the page does not need to be rewritten to disk.

**Once the REDO phase is finished**, the database is in the exact state that it was in when the crash occurred. The set of active transactions—called the `undo_set`—has been identified in the Transaction Table during the analysis phase.

Now, the UNDO **phase** proceeds by scanning backward from the end of the log and undoing the appropriate actions. A compensating log record is written for each action that is undone. The UNDO reads backward in the log until every action of the set of transactions in the `undo_set` has been undone. When this is completed, the recovery process is finished and normal processing can begin again.

**Example:**

Consider the recovery example shown in Figure 23.5. There are three transactions:  $T_1$ ,  $T_2$ , and  $T_3$ .  $T_1$  updates page  $C$ ,  $T_2$  updates pages  $B$  and  $C$ , and  $T_3$  updates page  $A$ .

(a)

Lsn	Last_lsn	Tran_id	Type	Page_id	Other_information
1	0	$T_1$	update	C	...
2	0	$T_2$	update	B	...
3	1	$T_1$	commit		...
4	begin checkpoint				
5	end checkpoint				
6	0	$T_3$	update	A	...
7	2	$T_2$	update	C	...
8	7	$T_2$	commit		...

(b)

TRANSACTION TABLE			DIRTY PAGE TABLE	
Transaction_id	Last_lsn	Status	Page_id	Lsn
$T_1$	3	commit	C	1
$T_2$	2	in progress	B	2

TRANSACTION TABLE			DIRTY PAGE TABLE	
Transaction_id	Last_lsn	Status	Page_id	Lsn
$T_1$	3	commit	C	1
$T_2$	2	in progress	B	2

TRANSACTION TABLE			DIRTY PAGE TABLE	
Transaction_id	Last_lsn	Status	Page_id	Lsn
$T_1$	3	commit	C	1
$T_2$	8	commit	B	2
$T_3$	6	in progress	A	6

**Figure 23.5**  
 An example of recovery in ARIES. (a) The log at point of crash. (b)  
 The Transaction and Dirty Page Tables at time of checkpoint. (c)  
 The Transaction and Dirty Page Tables after the analysis phase.

Figure 23.5(a) shows the partial contents of the log, and Figure 23.5(b) shows the contents of the Transaction Table and Dirty Page Table. Now, suppose that a crash occurs at this point. Since a checkpoint has occurred, the address of the associated begin\_checkpoint record is retrieved, which is location 4. **The analysis phase starts from location 4 until it reaches the end.** The end\_checkpoint record would contain the Transaction Table and Dirty Page Table in Figure 23.5(b), and the analysis phase will further reconstruct these tables. When the analysis phase encounters log record 6, a new entry for transaction  $T_3$  is made in the Transaction Table and a new entry for page A is made in the Dirty Page Table. After log record 8 is analyzed, the status of transaction  $T_2$  is changed to committed in the Transaction Table. Figure 23.5(c) shows the two tables after the analysis phase.

**For the REDO phase, the smallest LSN in the Dirty Page Table is 1. Hence the REDO will start at log record 1 and proceed with the REDO of updates.** The LSNs {1, 2, 6, 7} corresponding to the updates for pages C, B, A, and C, respectively, are not less than the LSNs of those pages (as shown in the Dirty Page Table). So those data pages will be read again and the updates reapplied from the log (assuming the actual LSNs stored on those data pages are less than the corresponding log entry). At this point, the REDO phase is finished and the **UNDO phase starts.** From the Transaction Table (Figure 23.5(c)), **UNDO is applied only to the active transaction  $T_3$ .** The UNDO phase starts at log entry 6 (the last update for  $T_3$ ) and proceeds backward in the log. The backward chain of updates for transaction  $T_3$  (only log record 6 in this example) is followed and undone.

## UNIT 5

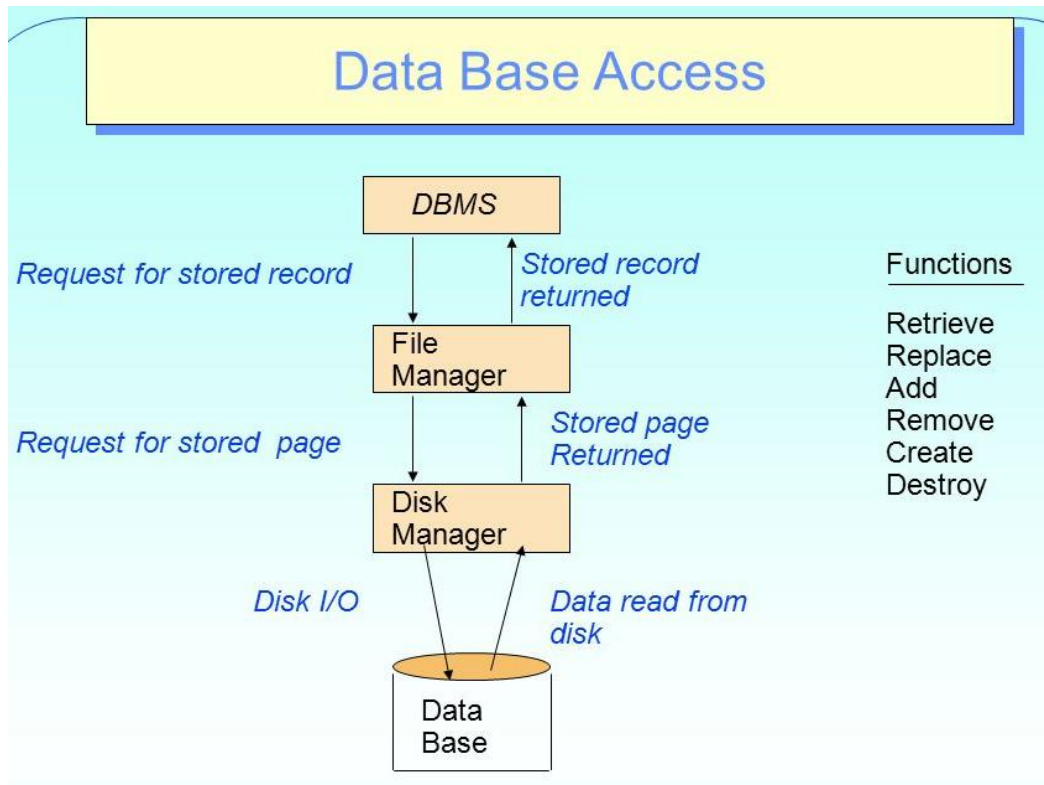
## OVERVIEW OF STORAGE AND INDEXING

**Unit 5 Contents at a glance:**

1. Data on external storage,
2. file organizations
3. indexing,
4. index data structures,
5. comparison of file organizations,
6. RAID
7. Tree structured indexing -intuition for tree indexes,
8. indexed sequential access method (ISAM),
9. B+ Trees -a dynamic tree structure.

**1. Data on external storage:**

- Data in a DBMS is stored on storage devices such as disks and tapes
- The disk space manager is responsible for keeping track of available disk space.
- The file manager, which provides the abstraction of a file of records to higher levels of DBMS code, issues requests to the disk space manager to obtain and relinquish space on disk.

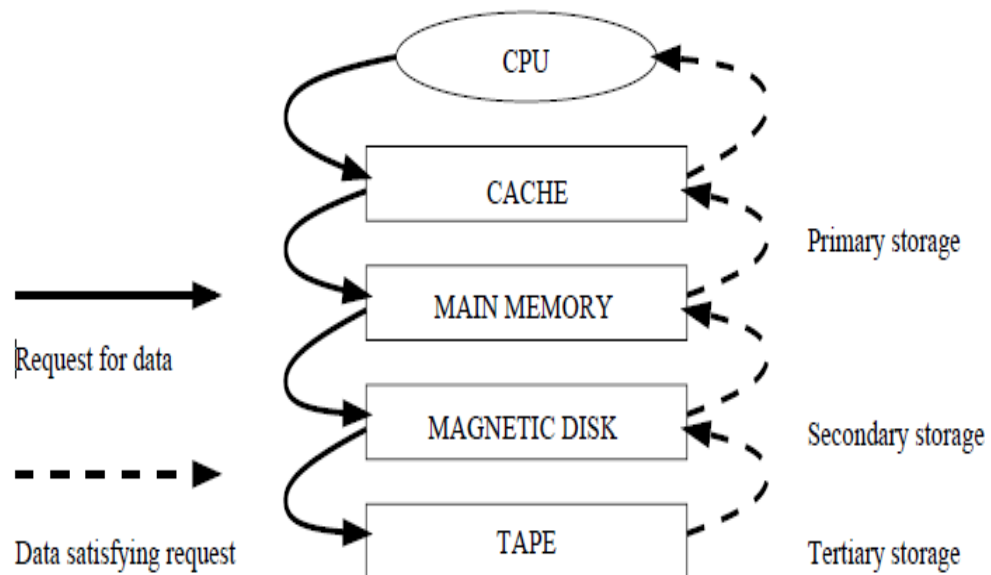


### Storage Manager Component :

A Storage Manager is a component or program module that provides the interface between the low-level data stored in the database and the application programs/queries submitted to the system. The Storage Manager Components include –

1. **File Manager**- File manager manages the file space and it takes care of the structure of the file. It manages the allocation space on disk storage and the data structures used to represent info stored on other media.
2. **Buffer Manager** – It transfers blocks between disk (or other devices) and Main Memory. A DMA (Direct Memory Access) is a form of Input/Output that controls the exchange of blocks process. When a processor receives a request for a transfer of a block, it sends it to the DMA Controller which transfers the block uninterrupted.
3. **Authorization and Integrity Manager** – This Component of storage manager checks for the authority of the users to access and modify information, as well as integrity constraints (keys, etc).
4. **Disk Manager**- The block requested by the file manager is transferred by the Disk Manager.

### Memory Hierarchy:



**Figure: Memory Hierarchy**

At the top, we have primary storage, which consists of cache and main memory, and provides very fast access to data. Then comes secondary storage, which consists of slower devices such as magnetic disks. Tertiary storage is the slowest class of storage devices; for example, optical disks and tapes.

**Primary Storage:**

1. all the primary storage level, the memory hierarchy includes at the most expensive end cache memory, which is a static RAM (Random Access Memory ) cache memory is mainly used by the CPU to speedup execution programs.
2. the next level of primary storage is DRAM (Dynamic Random Access Memory ), which provides the main work area for the CPU for keeping programs and data , which is popularly called as main memory .
3. the advantages of DRAM is its low cost, which continuous to decrease ; the drawback is its volatility and lower speed compared with static RAM.

**Secondary Storage:**

At the secondary storage level, the hierarchy includes magnetic disks, as well storage in the form of CD - ROM (Compact Disk - Read Only Memory ) devices.

Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.

**Tertiary storage:**

At the tertiary storage level, the hierarchy includes optical disks and tapes as the least expensive end.

The storage capacity anywhere in the hierarchy is measured in kilobytes (k bytes or bytes), megabytes (M bytes or 1 million bytes), gigabytes (G byte or billion bytes), and even terabytes (1000 G bytes).

**Explanation:****DRAM:**

programs reside execute in **DRAM** . Generally, large permanent database reside on secondary storage, and portions of the database are read into and written from buffers is main memory as needed. personal computers and work stations have tens of megabytes of data in DRAM. it is become possible to load a large fraction of the database into main memory. an example is telephone switching applications, which store databases that contain routing and line information in main memory.

**Flash Memory:**

1. Between **DRAM and magnetic disk storage**, another form of memory resides, **flash memory**, which is becoming common, particularly because it is non - volatile.
2. flash memories are high density, high - performance memories using EEPROM (Electrically Erasable programmable Read -only Memory) technology.
3. the advantage of flash memory is the fast access speed;
4. the disadvantage is that an entire block must be erased and written over at a time.

**Magnetic disk storage:**

- primary medium for long-term storage.
- Typically the entire database is stored on disk.
- Data must be moved from disk to main memory in order for the data to be operated on.
- After operations are performed, data must be copied back to disk if any changes were made.
- Disk storage is called **direct access** storage as it is possible to read data on the disk in any order (unlike sequential access).
- Disk storage usually survives power failures and system crashes.

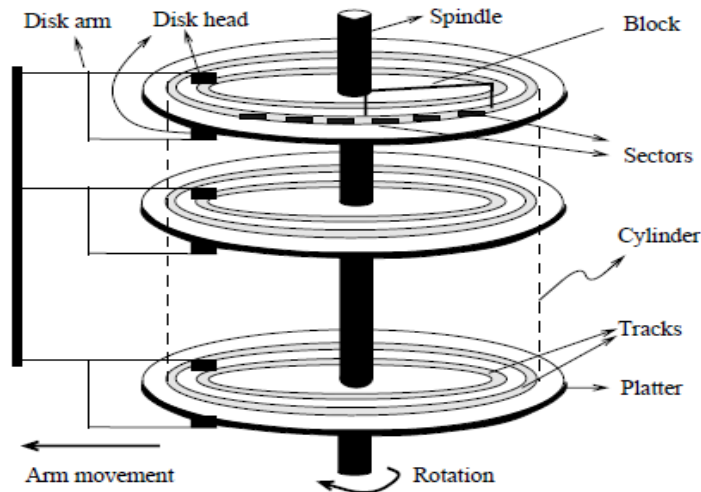


Figure 7.2 Structure of a Disk

**Figure: Structure of magnetic disk**

**Access time:** the time it takes from when a read or write request is issued to when data transfer begins.

**Data-transfer rate**– the rate at which data can be retrieved from or stored to the disk.

**Mean time to failure (MTTF)**– the average time the disk is expected to run continuously without any failure.

**CD-ROM:**

CD - ROM disks store data optically and are read by a laser. CD - ROM s contain pre - recorded data that cannot be overwritten. WORM (Write - Once - Read - Many disks) are a form of optical storage used for archiving data; they allow data to be written once and read any number of times without the possibility of erasing. the DVD (Digital Video Disks) is a recent standard for optical disks allowing fourteen to fifteen gigabytes of storage per disks.

**Tapes:**

1. Tapes are relatively not expensive and can store very large amount of data. when we maintain data for a long period but do expect to access it very often.
2. used primarily for backup and archival data.
3. Cheaper, but much slower access, since tape must be read sequentially from the beginning.
4. Used as protection from disk failures!
5. A Quantum DLT 4000 drive is a typical tape device; it stores 20 GB of data and can store about twice as much by compressing the data.

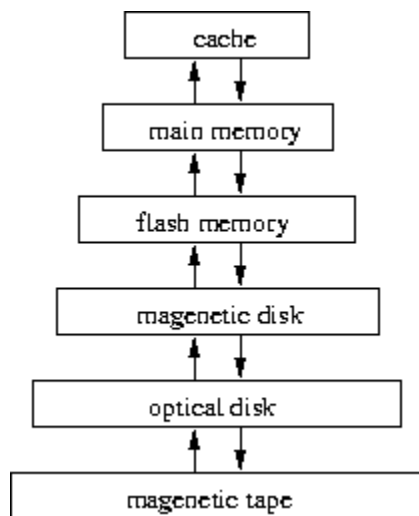


Figure: storage device hierarchy

## 2. File Organizations:

Storing the files in certain order is called file organization. The main objective of file organization is

- Optimal selection of records i.e.; records should be accessed as fast as possible.
- Any insert, update or delete transaction on records should be easy, quick and should not harm other records.
- No duplicate records should be induced as a result of insert, update or delete
- Records should be stored efficiently so that cost of storage is minimal.

Some of the file organizations are

1. Sequential File Organization
2. Heap File Organization
3. Hash/Direct File Organization
4. Indexed Sequential Access Method
5. B+ Tree File Organization
6. Cluster File Organization

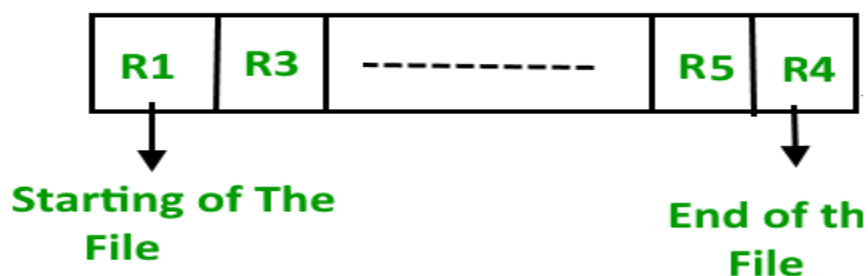
### 1. Sequential File Organization:

In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key.

The easiest method for file Organization is Sequential method. In this method the the file are stored one after another in a sequential manner. There are two ways to implement this method:

1. Pile File Method
2. Sorted File

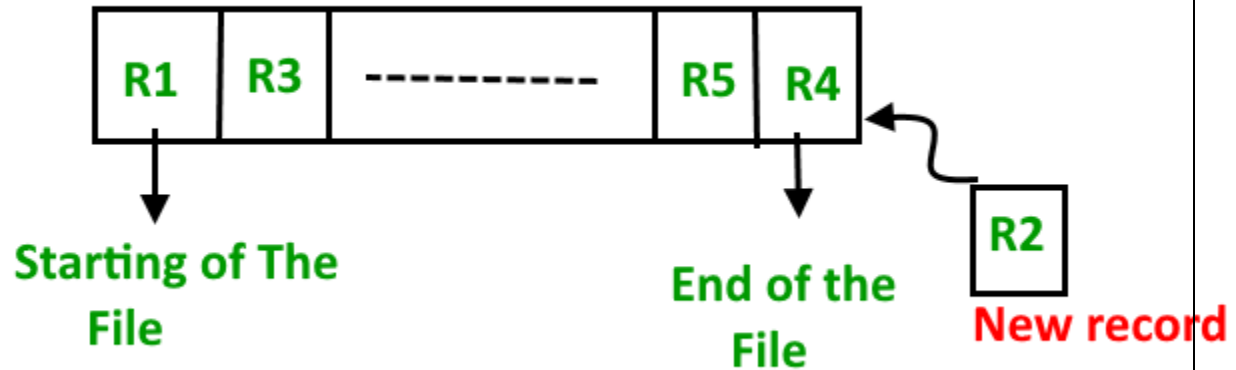
1. **Pile File Method** – This method is quite simple, in which we store the records in a sequence i.e one after other in the order in which they are inserted into the tables.



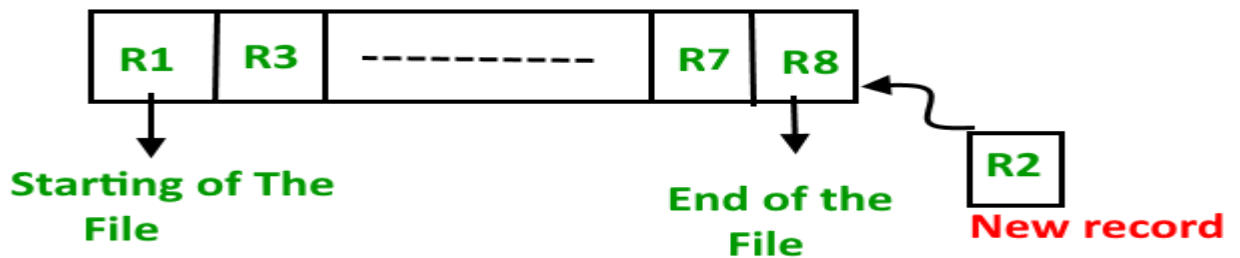


**Insertion of new record –**

Let the R1, R3 and so on upto R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.

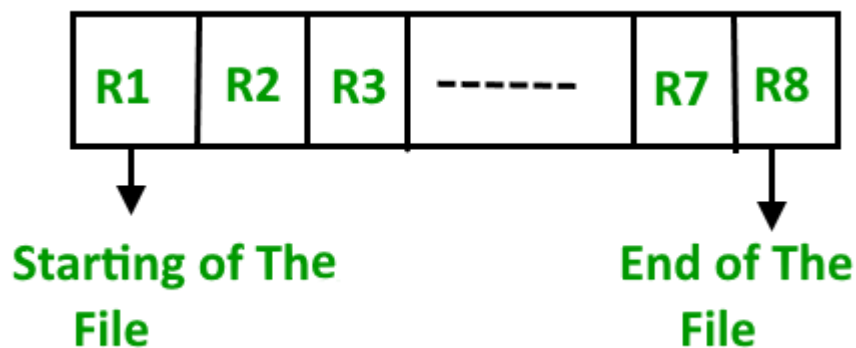


2. **Sorted File Method** –In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. Sorting of records may be based on any primary key or any other key.



**Insertion of new record –**

Let us assume that there is a preexisting sorted sequence of four records R1, R3, and so on upto R7 and R8. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence .



**Pros and Cons of Sequential File Organization –****Pros –**

- Fast and efficient method for huge amount of data.
- Simple design.
- Files can be easily stored in magnetic tapes i.e cheaper storage mechanism.

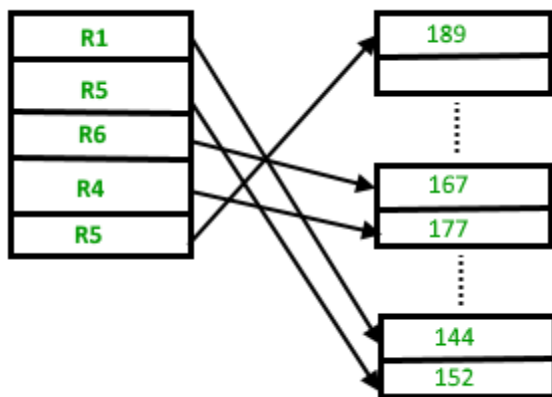
**Cons –**

- Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
- Sorted file method is inefficient as it takes time and space for sorting records.

**2. Heap File Organization:**

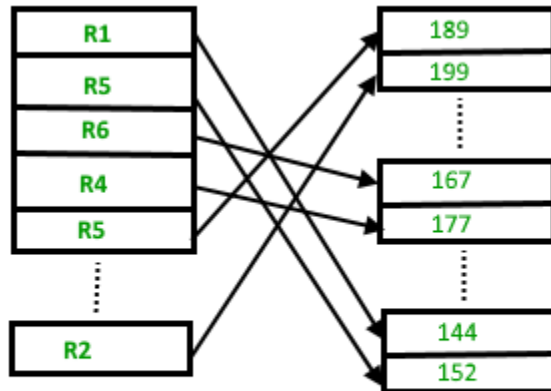
When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area.

Heap File Organization works with data blocks. In this method records are inserted at the end of the file, into the data blocks. No Sorting or Ordering is required in this method. If a data block is full, the new record is stored in some other block, Here the other data block need not be the very next data block, but it can be any block in the memory. It is the responsibility of DBMS to store and manage the new records.



**Insertion of new record –**

Suppose we have four records in the heap R1, R5, R6, R4 and R3 and suppose a new record R2 has to be inserted in the heap then, since the last data block i.e data block 3 is full it will be inserted in any of the database selected by the DBMS, lets say data block 1.



If we want to search, delete or update data in heap file Organization then we will traverse the data from the beginning of the file till we get the requested record. Thus if the database is very huge, searching, deleting or updating the record will take a lot of time.

**Pros and Cons of Heap File Organization –****Pros –**

- Fetching and retrieving records is faster than sequential record but only in case of small databases.
- When there is a huge number of data needs to be loaded into the database at a time, then this method of file Organization is best suited.

**Cons –**

- Problem of unused memory blocks.
- Inefficient for larger databases.

### 3. Hash File Organization:

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

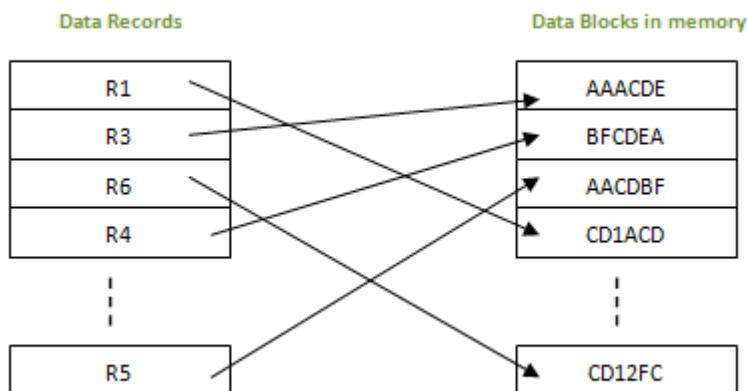
In this method of file organization, hash function is used to calculate the address of the block to store the records.

The hash function can be any simple or complex mathematical function.

The hash function is applied on some columns/attributes – either key or non-key columns to get the block address.

Hence each record is stored randomly irrespective of the order they come. Hence this method is also known as Direct or Random file organization.

If the hash function is generated on key column, then that column is called hash key, and if hash function is generated on non-key column, then the column is hash column.



When a record has to be retrieved, based on the hash key column, the address is generated and directly from that address whole record is retrieved. Here no effort to traverse through whole file. Similarly when a new record has to be inserted, the address is generated by hash key and record is directly inserted. Same is the case with update and delete.

#### ***Advantages of Hash File Organization***

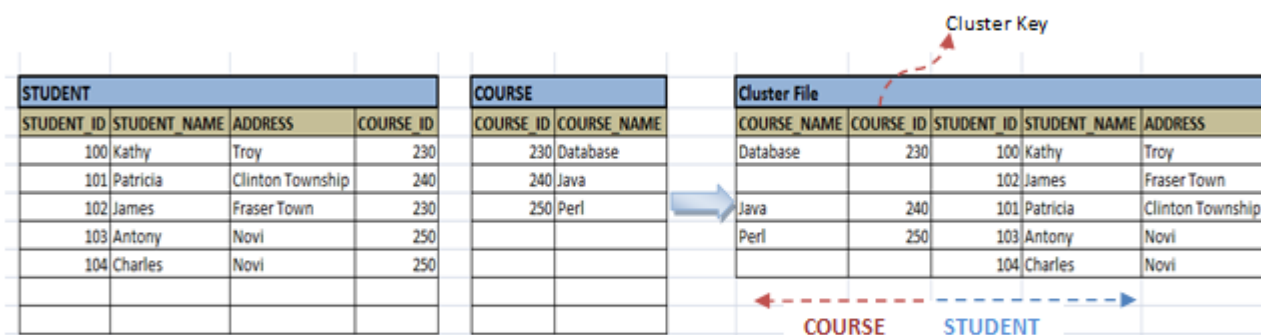
- Records need not be sorted after any of the transaction. Hence the effort of sorting is reduced in this method.
- Since block address is known by hash function, accessing any record is very faster. Similarly updating or deleting a record is also very quick.

- This method can handle multiple transactions as each record is independent of other. i.e.; since there is no dependency on storage location for each record, multiple records can be accessed at the same time.
- It is suitable for online transaction systems like online banking, ticket booking system etc.

**clustered file organization:**

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

In this method two or more table which are frequently used to join and get the results are stored in the same file called clusters. These files will have two or more tables in the same data block and the key columns which map these tables are stored only once. This method hence reduces the cost of searching for various records in different files. All the records are found at one place and hence making search efficient.



## REDUNDANT ARRAY OF INDEPENDENT DISKS(RAID)

RAID or **Redundant Array of Independent Disks**, is a technology to connect multiple secondary storage devices and use them as a single storage media.

RAID consists of an array of disks in which multiple disks are connected together to achieve different goals. RAID levels define the use of disk arrays.

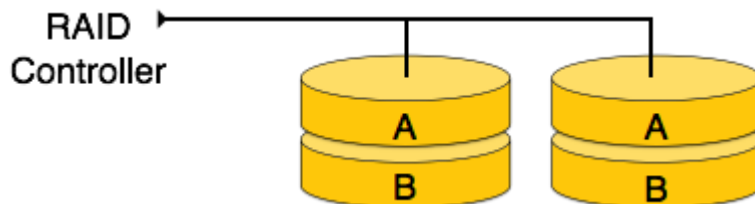
### RAID 0

In this level, a striped array of disks is implemented. The data is broken down into blocks and the blocks are distributed among disks. Each disk receives a block of data to write/read in parallel. It enhances the speed and performance of the storage device. There is no parity and backup in Level 0.



### RAID 1

RAID 1 uses mirroring techniques. When data is sent to a RAID controller, it sends a copy of data to all the disks in the array. RAID level 1 is also called **mirroring** and provides 100% redundancy in case of a failure.



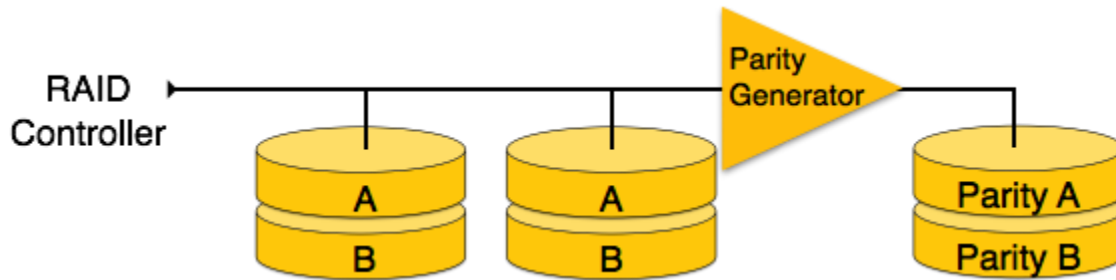
### RAID 2

RAID 2 records Error Correction Code using Hamming distance for its data, striped on different disks. Like level 0, each data bit in a word is recorded on a separate disk and ECC codes of the data words are stored on a different set disks. Due to its complex structure and high cost, RAID 2 is not commercially available.



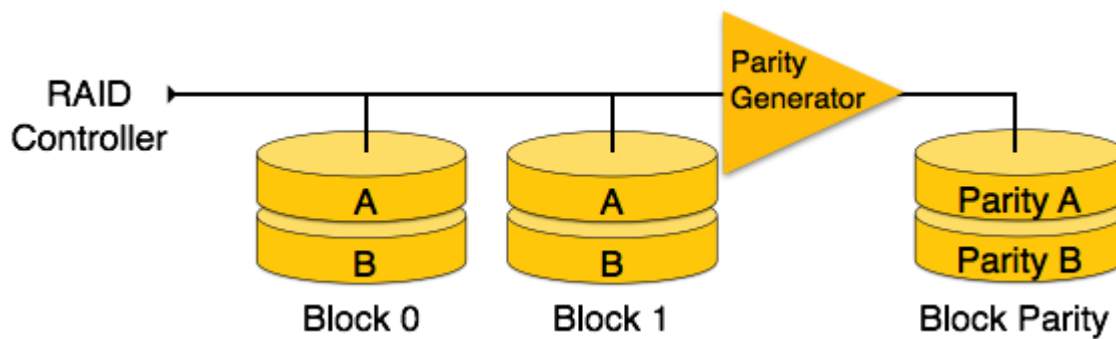
**RAID 3**

RAID 3 stripes the data onto multiple disks. The parity bit generated for data word is stored on a different disk. This technique makes it to overcome single disk failures.



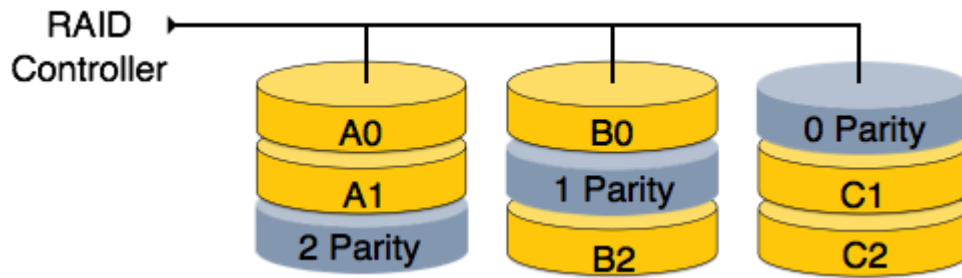
**RAID 4**

In this level, an entire block of data is written onto data disks and then the parity is generated and stored on a different disk. Note that level 3 uses byte-level striping, whereas level 4 uses block-level striping. Both level 3 and level 4 require at least three disks to implement RAID.



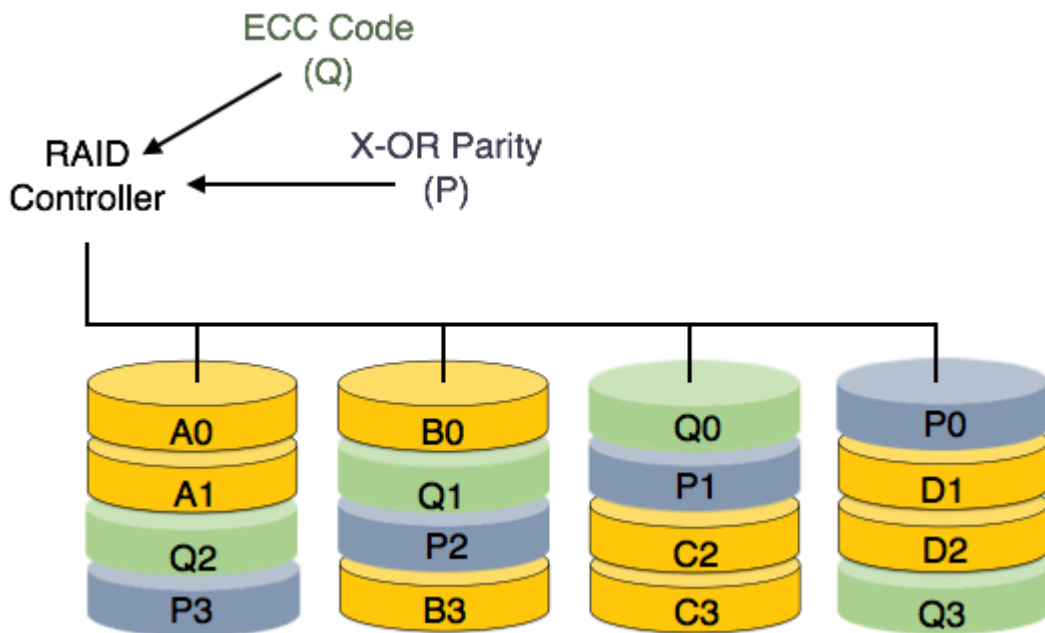
**RAID 5**

RAID 5 writes whole data blocks onto different disks, but the parity bits generated for data block stripe are distributed among all the data disks rather than storing them on a different dedicated disk.



**RAID 6**

RAID 6 is an extension of level 5. In this level, two independent parities are generated and stored in distributed fashion among multiple disks. Two parities provide additional fault tolerance. This level requires at least four disk drives to implement RAID.





**Comparison of file organizations:**

The operations to be considered for comparisons of file organizations are below:

- **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 the pool).
- **Search with equality selection:** Fetch all records that satisfy an equality selection, for example, “Find the Students record for the student with *sid* 23.” 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, for example, “Find all Students records with *name* alphabetically after ‘Smith.’ ”
- **Insert:** Insert a given record into the 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 the file organization, we may have to fetch, modify, and write back other pages as well.
- **Delete:** 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.

<i>File Type</i>	<i>Scan</i>	<i>Equality Search</i>	<i>Range Search</i>	<i>Insert</i>	<i>Delete</i>
Heap	$BD$	$0.5BD$	$BD$	$2D$	$Search + D$
Sorted	$BD$	$D \log_2 B$	$D \log_2 B + \# \text{ matches}$	$Search + BD$	$Search + BD$
Hashed	$1.25BD$	$D$	$1.25BD$	$2D$	$Search + D$

Figure 8.1 A Comparison of I/O Costs

**B - number of data pages**

**R records per page**

**D- average time to read or write a disk page**

**C- average time to process a record**

**Indexing:**

**Index:**

**Dense Index:**

**Sparse Index:**

**Primary indexing:**

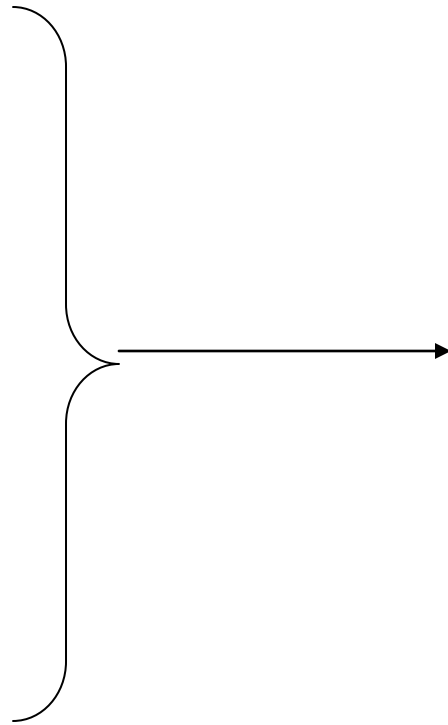
**secondary indexing:**

**cluster indexing:**

**multi value indexing:**

**ISAM**

**B+ trees:**



Read from class notes