MODULE-I

Data

- Datum: Singular form of data
- Data: Plural form of a datum
- Data is the known facts that can be recorded and have implicit meaning, which can be represented by symbols
- Examples of data:

  Weights, Prices, costs, number of item sold, employee names, product names, addresses, tax codes, registration marks of a student, roll numbers, percentage etc.

Information

- It is data that has been converted into a more useful or intelligible form

  or
- It is the set of data arranged in an ordered and useful form

  or
- Processed data is known as the information
- Examples : Time table, Merit List, Report card ,Pay-slips, Schedules, reports , worksheet, bar charts etc.

Difference between Data and information

- Data is the material on which computer programs work upon, but data themselves have no meaning
- Example: The sequence of digits 240343 is meaningless by itself since it could refer to a data of birth, a part number of for a automobile, the number of rupees spent on a project, population of a town, the number of people employed in a large organization etc.
- Once we know that the sequence refers to (say 24-03-43 means date of birth), then it becomes meaningful and can be called information
- A set of words would be data but text would be information
- Example: "Annual examination, Amitabh, Jyotsna, Physics" is a set of data and " Jyotsna scored the highest marks in physics in Annual examination" is information

Need of information

Information is needed to

- Gain knowledge about the surroundings, and whatever is happening in the society and universe
- Keep the system up-to-date
- Know about the rules and regulation

Database:

- Definitions (6 types) of database are:
  - The related information when placed in an organized form makes a database
  - A database is a collection of related data

- It is the shared collection of logically related data (and a description of this data), designed to meet the information needs of an organization.
- A database is a collection of stored operational data used by application systems
- A database can be defined as a collection of files, file is a collection of records, record is a collection of fields and field is a memory used to store a unit of data item
- The database is an integrated collection of related files

- System catalog (metadata- data about data) provides description of data to enable program–data independence.
- Logically related data comprises entities, attributes, and relationships of an organization's information.

Characteristics of data in a database

- Shared: Data in a database are shared among different users and applications
- Persistence: data in a database exist permanently in the sense that the data can live beyond the scope of the process that created it
- Validity/Integrity/Correctness: Data should be correct with respect to the real world entity that they present
- Security: Data should be protected from unauthorized access
- Consistency: Whenever more than one data element in a database represents related real world values, the values should be consistent with respect to the relationship
- Non-redundancy: No two data items in a database should represent the same real world entity
- Independence : Data at different levels should be independent of each other so that the changes in the one level should not affect the other levels

What do we do with the database?

- To add new information
- To view or retrieve the stored information
- To modify or edit the existing information
- To remove or delete the unwanted information
- To arrange the information in a desired order

Database and Computers

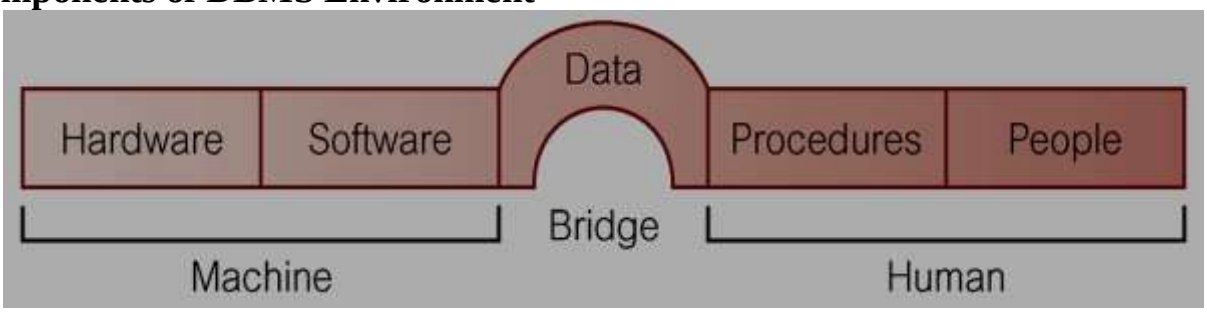
- Computerized database is more suited than manual database because of the following reasons:
  - Computer has a large capacity and can store thousands of records at a time
  - It has high speed, within no time it searches any desired information , arrange the data in alphabetic order, do calculations on the data and make repetitions and so on
  - Computer is more accurate
  - Data in computers can be stored in the form of a file, records and fields

Database Management System (DBMS)

- It is a software system that enables users to define, create, and maintain the database and provides controlled access to this database.
- DBMS is a collection of programs that enables users to create and maintain a database
- **DBMS is a general –purpose software system that facilitates the processes of defining, constructing , manipulating, sharing, protecting and maintaining the databases among various users and applications**

- Defining a database involves specifying the data types, structures, and constraints for the data to be stored in the database
- Constructing the database is the process of storing the data itself on some storage medium that is controlled by DBMS
- Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the mini world, and generating reports from the data
- Sharing a database allows multiple users and programs to access the database concurrently
- Protection includes both system protection against h/w or s/w malfunction (or crashes) and against unauthorized or malicious access
- Maintenance : A typical large database may have a life cycle of many years, so that DBMS must be able to maintain the database system by allowing the system to evolve as requirements change over time

Components of DBMS Environment



- **Hardware**
  - **Can range from a PC to a network of computers.**
- **Software**
  - **DBMS, operating system, network software (if necessary) and also the application programs.**
- **Data**
  - **Used by the organization and a description of this data called the schema.**

- **Procedures**
  - **Instructions and rules that should be applied to the design and use of the database and DBMS.**
- **People**

Data dictionary(or data directory or system catalog) and meta data:

- **The result of compilation of DDL statements is a set of tables that is stored in a special file called data dictionary or data directory**
- **A data dictionary is a file that contains meta data(i.e. data about data).**
- **The meta data contains definitions of records, data items, and other objects that are of interest to users or are required by the DBMS**

Database system

- Database system = Database + DBMS S/W

Basic services to the user provided by DBMS

- Database creation
- Database maintenance
- Database processing

Functions of a database system

- Data storage, retrieval, update

- A user-accessible catalog
- Transaction support
- Concurrency control
- Recovery services
- Authentication services
- Integrity services
- Data independence
- Utility services

Requirements of a Database system

- Efficiency
- Resilience
- Access control
- Persistence

History of Database Systems

- First-generation
  - Hierarchical and Network
- Second generation
  - Relational
- Third generation
  - Object Relational
  - Object-Oriented

Examples of Database Applications

- Purchases from the supermarket
- Purchases using your credit card
- Booking a holiday at the travel agents
- Using the local library
- Taking out insurance
- Using the Internet
- Studying at university

Following two approaches are used for storing data in computers

- File-based approach
- Database approach

File-Based Systems- Traditional approach

- Collection of application programs that perform services for the end users (e.g. reports).
- Each program defines and manages its own data.
- In traditional file based systems, the data is input to the computer to get certain output
- Each program or application is designed in isolation in the sense that it has its own set of data files
- There is every chance that the same data could be present in more than one files pertaining to same or different applications
  - Example: The data of a student (name, roll no etc.) could be present in both in attendance file and fee collection file. Such a situation leads to data redundancy and wasted storage space
- The data stored in different files has to be accessed through different programs or reports. This leads to data isolation
- Extraction of information is cumbersome and time consuming in the file based system

Limitations of File-Based Approach

- Separation and isolation of data
- Duplication of data (data redundancy)
- Data dependence

- Incompatible file formats
- Fixed Queries/Proliferation of application programs
- Inconsistent data
- Integrity problems
- Atomicity problems
- Concurrent access anomalies
- Security problems
- Data inflexibility
- Difficulty in accessing data
- Difficulty in representing data from the **user's view**

- **Separation and isolation of data**
  - Each program maintains its own set of data.
  - Users of one program may be unaware of potentially useful data held by other programs.
  - When the data is stored in separate files it becomes difficult to access. It becomes extremely complex when the data has to be retrieved from more than two files as a large amount of data has to be searched

- **Duplication of data**
  - Same data is held by different programs.
  - Wasted space and potentially different values and/or different formats for the same item.
  - Duplication of data costs time and money
  - Duplication can lead to loss of data integrity i.e the data is no longer consistent
    - Example: Consider the duplication of data between the payroll and personnel departments . If a member of staff moves to new house and change of address is communicated only to personnel and not to payroll, the person's pay slip will sent to the wrong address
- **Data dependence**
  - File structure is defined in the program code.
  - The physical structure and storage of data files and records are defined in the application code, which means that it is extremely difficult to make changes to the existing structure
  - The programmer would have to identify all the affected programs, modify them and reset them. This characteristics of the file based system is called program data dependence
- **Incompatible file formats**
  - Programs are written in different languages, and so cannot easily access each other's files.
  - Structure of files is embedded in the application programs
  - Structures are dependent on the application programming language
    - Example: The structure of a file generated by a COBOL program may be different from the structure of a file generated by a C program. The direct incompatibility of such files makes them difficult to process jointly
- **Fixed Queries/Proliferation of application programs**
  - Programs are written to satisfy particular functions.
  - Any new requirement needs a new program.
  - Any query or report needed by the organization has to be developed by the application programmer

- Producing different types of queries or reports is not possible in file based systems, as a result in some organizations the type of queries or reports to be produced is fixed and no new query or report of the data could be generated

- Inconsistent data: The data in a file system can become inconsistent if more than one person modifies the data concurrently
  - Example: If any student changes the residence and change is notified to only his/her file and not to bus list. Entering wrong data is also another reason for inconsistencies
- Integrity problems: The data values stored in the database must satisfy certain types consistency constraints
  - Example: The balance of a bank account may fall below a prescribed amount(say, $25). Developers enforce these constraints in the system in the system by adding appropriate code in the application programs. However , when new constraints are added , it is difficult to change the programs to enforce
- Atomicity problems: A computer system, like any other mechanical or electronically device , is subject to failure.
  - Example: Consider a program to transfer $50 from A/C A to B. If a system failure occurs during the execution of the program, it is possible that the $ 50 was removed from A/C A but was not credited to A/C B, resulting in an inconsistent database state. But funds transfer must be atomic- it must happen in its entirely or not at all. It is difficult to ensure this property in a conventional file-processing system
- **Concurrent access anomalies:** Interaction of concurrent updates may result in inconsistent data
  - Example: Consider bank A/C A, containing $500. If two customers withdraw funds(say $ 50 and $100) from A/C A at about the same time, the result of the concurrent executions may leave the A/C in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce the value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $500 and write back $450 and $400 respectively. Depending on which one writes the value last, the A/C may contain either $450 or $400, rather than the correct value of $350
- **Security problems**: Not every user of the database system should be able to access all the data
  - Example: In a banking system , payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer A/Cs
- **Data inflexibility:** Program data interdependency and data isolation , limited the flexibility of file processing systems in providing users with adhoc information requests
- **Difficulty in accessing data:**
- **Difficulty in representing data from the user's view :** To create useful applications for the user , often data from various files must be combined . In file processing it was difficult to determine relationships between isolated data in order to meet user requirements

Database Approach

- Arose because:
  - Definition of data was embedded in application programs, rather than being stored separately and independently.

- No control over access and manipulation of data beyond that imposed by application programs.

- Result:
  - The database and Database Management System (DBMS).

Advantages of DBMS

- Control of data redundancy
- Data consistency
- More information from the same amount of data
- Sharing of data
- Improved data integrity
- Improved security
- Enforcement of standards
- Economy of scale
- Balanced conflicting requirements
- Improved data accessibility and responsiveness
- Increased productivity
- Improved maintenance through data independence
- Increased concurrency
- Improved backup and recovery services

Control of data redundancy: In file system, each application ha its own private files, which can't be shared between multiple applications. This can lead t o considerable redundancy in the stored data, which results in wastage of storage space. By having centralized database most of this an be avoided. It is not possible that all redundancy should be eliminated.

Data consistency: When the same data is duplicated and changes are made at one site, which is not propagated to the other site, it gives rise to inconsistency and the two entries regarding the same data will not agree. At such times the data is said to be inconsistent. So if the redundancy is removed chances of having inconsistent data is also removed.

Sharing of data :

Disadvantages of DBMS

- Complexity
- Size
- Cost of DBMS
- Additional hardware costs
- Cost of conversion
- Performance
- Higher impact of a failure

Complexity The provision of the functionality that is expected of a good DBMS makes the DBMS an extremely complex piece of software. Database designers, developers, database administrators and end-users must understand this functionality to take full advantage of it. Failure to understand the system can lead to bad design decisions, which can have serious consequences for an organization

Size The complexity and breadth of functionality makes the DBMS an extremely large piece of software , occupying many megabytes of disk space and requiring substantial amounts of memory to run efficiently

Performance: Typically, a file based system is written for a specific application, such as invoicing. As a result , performance is generally very good. However , the DBMS is written to be more general, to cater for many applications rather than just one. The effect is that some applications may not run as fast as they used to.

Higher impact of a failure: The centralization of resources increases the vulnerability of the system. Since all users and applications rely on the availability of the DBMS, the failure of any component can bring operations to halt.

Cost of DBMS: The cost of DBMS varies significantly, depending on the environment and functionality provided. There is also the recent annual maintenance cost.

Additional hardware costs: Te disk storage requirements for the DBMS and the database may necessitate the purchase of additional storage space. Furthermore, to achieve the required performance it may be necessary to purchase a larger machine, perhaps even a machine dedicated to running the DBMS. The procurement of additional hardware results in further expenditure

Cost of conversion: In some situations, the cost of the DBMS and extra hardware may be insignificant compared with the cost of converting existing applications to run on the new DBMS and hardware. This cost also includes the cost of training staff to use these new systems and possibly the employment of specialist staff to help with conversion and running of the system

Data models

- Model
- Data model
- Components of a data model
- Purpose of data model
- Categories of data model

Model

A model in database system basically defines the structure or organization of data and a set of

operations on that data

Data Model

- How is the data organized in a database ?
- A database model defines
  - the logical data structure
  - data relationships
  - data relationship constraints
- A data model is an integrated collection of concepts for
  - describing the data
  - manipulating data
  - relationships between data
  - and constraints on the data in an organization

Components of a data model

- A structural part
- A manipulative part
- A set of integrity rules

Structural part

Consists of a set of rules according to which databases can be constructed

A manipulative part

Defines the types of operation(updating ,retrieving and changing the structure) that are

allowed on the data type

A set of integrity rules

Ensures that the data are accurate

Purpose of a data model

To represent data and to make the data understandable

Categories of data models

- Record based data model
  (External/ Representational/ Implementation data model)
- Object-based data model
  (Conceptual/ Logical/ high-level data model)
- Physical data model
  (Internal/Low-level data model)

Record based data models

- Use records as the key data representation components
- Represent data by using record structures
- Used in describing data at the logical and view levels
- Used to specify the overall logical structure of the database
- Provides a higher-level description of the implementation
- The database is structured in fixed format records of several types
- Each record type defines a fixed number of fields (or attributes)
- Each field is usually of fixed length

Kinds of Record based data models

- Hierarchical data model
- Network data model
- Relational data model

Hierarchical data model (1950)

- Data and relationships among data are represented by records and links, respectively, through hierarchy of data values
- Data can be represented as an ordered tree in the sense that it consists of data or files with parent and child relationships with the restriction being that a child can have onl one parent
- Represents one-to-many relationships between a parent and its children in the form of hierarchical tree

Advantages of hierarchical data model

- Simple and easy to use
- Data with hierarchical relationships can be naturally mapped on this model
- Suitable for applications having one-to-many(1:N) relationships
  e.g. Employee Department, Bill of materials
- Data security
- Data integrity(Child segments are always automatically referred by it's parent)
- Efficiency(for database containing a large number of 1:N (one-to-many) relationships)

Disadvantages of Hierarchical data model

- Suffers from the insertion, update and deletion anomalies (operational anomalies)
- Retrieval operation is complex and asymmetric (operational anomalies)
- Non-hierarchical relationships are difficult to map on this model. Thus it is inflexible
- The processing is sequential along the branches of the tree and therefore the access time become longer
- The hierarchical tree is implemented through pointers from parents to their children. This activity require extra storage
- Deletion of parent deletes it's children nodes

- Changes in relationship require changes in the entire structure of the database
- Implementation Limitations: Many of the common relationships do not confirm to the 1:N format required by the hierarchical model.
- The many-to-many (N:N) relationships which are common in real life are very difficult to implement in this model

Examples of Hierarchical DBMS

- IBM's IMS(Information Management System)
- System 2000

Network data model(1969)(CODASYL Model)

- Data are represented by collection of records and relationships(sets) among data are represented by links, which can be viewed as pointers
- The records in the database are organized as collection of arbitrary graphs
- Allows 1:1(one-one), 1:M(one-many) & M:M (many-to-many) relationships among entities
- Deviates from hierarchical data model by allowing a child to have many parents

Advantages of Network data model

- Conceptual simplicity
- Capability to handle more relationship type like 1:1,1:M,M:M
- Ease of data access
- Data integrity
- Data independence
- Database standards: The network model is based on the standards formulated by the DBTG and augmented by ANSI/SPARC in 1970, which includes DDL & DML
- Doesn't suffer from insert, update and delete anomalies and retrieval operation is symmetric
  as compared to hierarchical data model

Disadvantages of Network model

- System complexity: Number of links tend to be extremely large as the complexity increases
- Operational anomalies: Network model's insertion, deletion and updation operations of any record require large number of pointer adjustments, which makes it's implementation very complex and complicated
- New queries can't be implemented
- Extra storage is required for pointers
- A high level language is needed to program the database
- General purpose query facility is not available
- Absence of structural independence

Examples of Network DBMS

- UNIVAC's DMS 1100
- DBMS 10-20 from DEC

Relational data model(1980)(Dr.E.F.Codd)

- Represents data as well as relationships among data in the form of tables
- Data is stored in the form of rows and columns in tables
- Rows of two tables can be put into association if they have common column characteristics .
  This is called relation.

Advantages of Relational model

- Structural independence
- Conceptual simplicity
- Design, implementation, maintenance ,& usage ease

- Ad hoc query capability
- Easy to understand (Since tabular structure is simple and familiar)
- Data manipulation is easy
- We can apply mathematical operations on tables
- Built in query language support on RDBMS, is available
- Very flexible data organization
- Doesn't suffer from insert, update and delete anomalies and retrieval operation is very simple and symmetric

Disadvantages of Relational model

- Hardware overheads
- Ease of design can lead to bad design
- Size of database becomes large
- ' Information is land' phenomenon

Examples of Relational data model

- Oracle
- Ingress
- Unity
- Sybase
- MS ACCESS
- SQL Server and so on.

Which data model to use?

- The data model that suits an organization depends on the following factors:
  - The organization's primary goals and requirements
  - The volume of daily transactions that will be done
  - The essential number of enquires that will be made by the organization
- The relational data model is widely preferred data model because:
  - It can be used for representing most of the real world objects and the relationships among them
  - Security and integrity are maintained easily
  - Increases the productivity of application programs
  - Relational tables show only the logical relationship
  - End users doesn't know the exact physical structure of a table
- The hierarchical and network data models are inherently unstable
  - because of the hidden pointers in the records
  - In the event of system errors, the chain addresses between the records could be damaged resulting in reduced data integrity
- Network model is not a preferred model due to it's complex nature
- Hierarchical model is useful only for those cases which are hierarchical in nature

Object-based data model

- The Entity relationship (E-R) model
- The object oriented model
- The semantic data model
- The functional data model

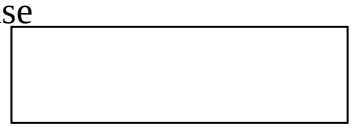
ER Model

- Useful in database design
- Mainly represented in graphical form using E-R diagrams
- It is the collection of real world objects called entities and their relationships

Some terms connected with ER model

- Entity:

– It is a “thing” or “object” in the real world that is distinguishable from other objects, that is described in the database
– Represented by rectangle symbol



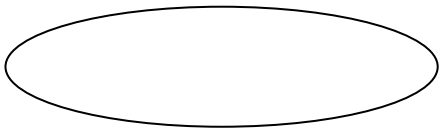
_ Examples

PERSON

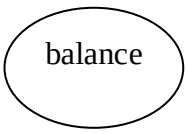
BANK ACCOUNT

Both “PERSON” and “BANK ACCOUNT” represent the entities

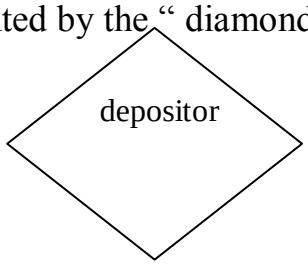
- Attribute
  - The property to describe an entity
  - Represented by an “ellipse” symbol
  -

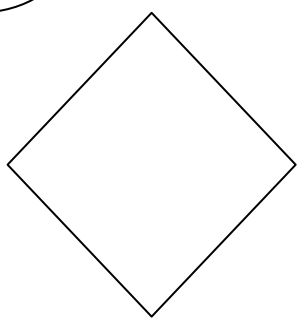


Ex. The entity “account” is described by the attributes “account number” and “ balance



- Relationship
  - An association among several entities
  - Represented by the “ diamond” symbol



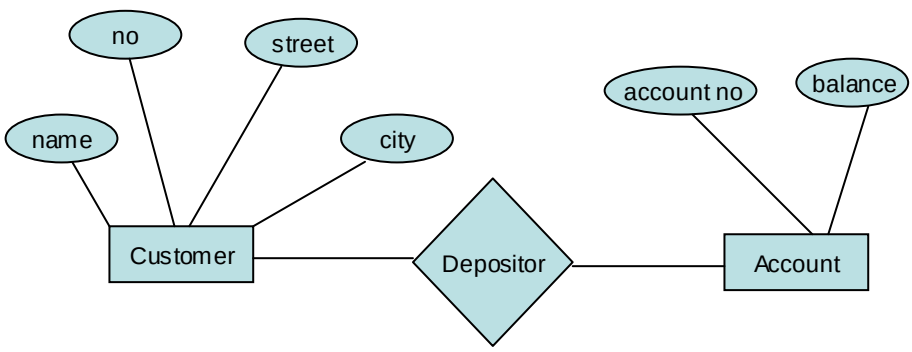


  - Example: A “depositor “ relationship associates a “customer” entity with each “ account” entity that he/she has

- Link
  - Line joining entities to attributes , entities to relationships and relationships to entities.

ER Diagram Example

- Example:



The object oriented data model

- Defines the database as a collection of objects that contains both data members/ values and operations that are allowed on the data
- The interrelationships and constraints are implemented through objects, links and message passing mechanisms
- Useful for databases where interrelationships are complex

Physical data model

- Describes how data is stored in the computer
- Representing information such as
  - Record structures
  - Record ordering
  - Access paths

Database system architecture

Some terms connected to database system architecture

- Schema (or Database Schema or Intension): The overall description of the database, which is specified during database design and is not expected to change frequently
- Schema diagram: A displayed schema is called a schema diagram
  Example: Schema diagram of "STUDENT" and "COURSE" are given below:

NAME	STUDENT NUMBER	CLASS	MAJOR

COURSE NAME	COURSE NUMBER	CREDIT HOURS	DEPARTMENTS

SCHEMA CONSTRUCT

- Each object in the schema is known as schema costruct
- Examples of Schema construct :
  ->Student
  ->Course

Database State(Or Snapshot or extension or instance)

- The data in the database at a particular moment in time is called as database state
- Example: The student construct contains the set of individual student entities(or records) as it's instance

Valid state

- The DBMS is partly responsible for ensuring that every state of the database is a valid state i.e. a state that satisfies the structure and constraints specified in the schema

Schema evolution

- Adding of new field to the schema construct for each record in a file

Database system architecture (Architecture of DBMS)

- The Logical DBMS architecture
- The physical DBMS architecture

Logical DBMS architecture

- The way data is stored and presented to users

The physical DBMS architecture

- Software components that make up a DBMS

Three-Schema/3-Level ANSI-SPARC architecture or logical DBMS architecture

- The database task group (DBTG)(appointed by the conference on data systems and languages(CODASYL,1971) recognized the need for a 2 level approach with system view called the schema and user's view called subschema

- ANSI-SPARC recognized the need for a 3 level approach with a system catalog in 1975, with the following levels:
    - External level or External schema or View level
    - Conceptual level or conceptual schema or logical level
    - Internal level or internal schema or physical

Objective of 3-level architecture

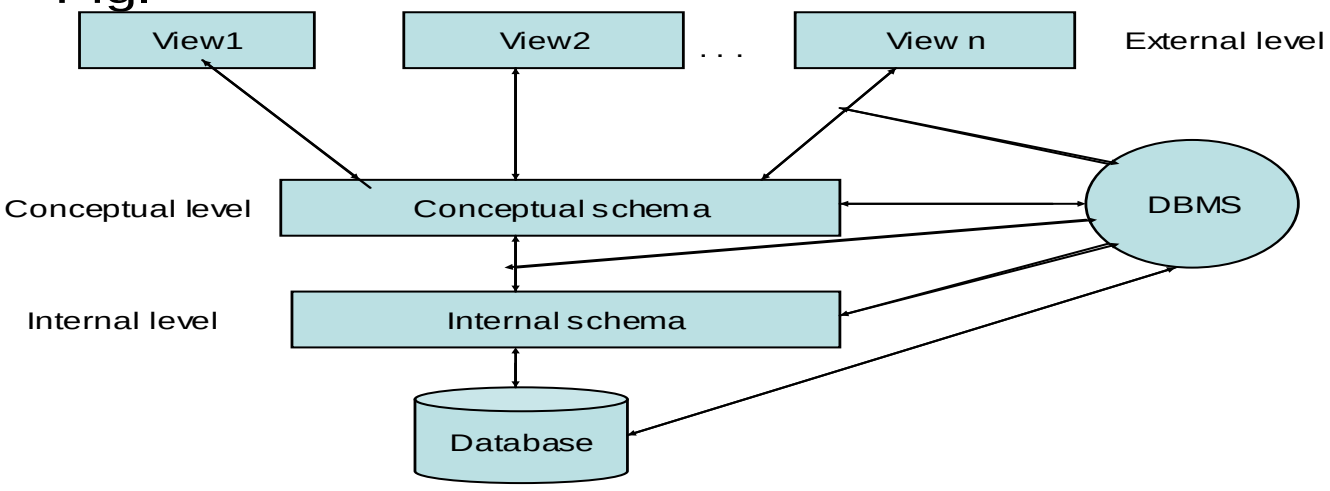
- Separating each user's view of the database from the way the database is physically represented

Reasons for the separation in 3-level database architecture

- Each user should be able to access the same data, but have a different customized view of the data
- Each user should be able change the way he or she views of the data, and this change should not affect other users.
- A user's interaction with the database should be independent of storage considerations
- The DBA should be able to change the database storage structures without affecting the user's views.
- The internal structure of the database should be unaffected by changes to the physical aspects of storage, such as the change over to a new storage device.
- The DBA should be able to change the conceptual structure of the database without affecting all users

Figure for 3-level architecture

- Fig.



External level

- User's view of the database
- Describes that part of the database that is relevant to each user.
- Insulates user's from the details of the internal and conceptual level.
- Uses the record based logical model to describe data at external level.

Conceptual level

- The community view of the database
- Hides the details of physical storage structures
- Describes what data is stored in the database and relationship among the data

- Contains the logical structure of the entire database as seen by the DBA.
- Represents
  - All entities, their attributes, and their relationships
  - The constraints on the data
  - Semantic information about the data
  - Security and integrity information
- Uses
  - Object based (E-R) data model and record based logical data models to describe the data at the conceptual level.

Internal level

- Physical representation of the database on the computer
- Describes how the data is stored in the database
- Concerned with
  - Storage space allocation for data and indexes
  - Record descriptions for storage
  - Record placements
  - Data compression and data encryption
- Covers the data structures and file organizations used to store data on storage devices
- Interfaces with O.S. access methods(file management techniques for storing and retrieving data records) to place the data on the storage devices, build the indexes, retrieve the data , and so on.
- Uses physical data model and describes the complete details of data storage and access paths for the database

Mapping

- Process of transforming requests and results between levels

Mapping between 3 different views

- External/Conceptual mapping
- Conceptual/Internal mapping

External/Conceptual mapping

- Each external schema is related to the conceptual schema by the external/conceptual mapping
- Correspondence among the records and the relationships of the external and conceptual views

Conceptual/Internal mapping

- Conceptual schema is related to the internal schema by the conceptual/ internal mapping
- Enables the DBMS to find the actual record or combination of records in physical storage that constitute a logical record in the conceptual schema, together with any constraints to be enforced on the operations for the logical record
- Allows any differences in entity names, attribute names, attribute order, data types, and so on , to be resolved.

Data abstraction

- System hides certain details of how the data are stored and maintained
- Provide users with an abstract view of the data
- 3-levels(View, logical & physical) of data abstraction
- A data model is a type of data abstraction that is used to provide the conceptual representation
- The characteristics that allows program-data independence and program-operation independence is called data abstraction

- Program- data independence is the property in which the structure of data files is stored in the DBMS catalog separately from the access programs. We call this property as **program -data independence.**
- In object-oriented and object-relational systems , users can define operations on data as part of the database definitions.
- An operation(also called a function or method) is specified in two parts.
- The interface(or signature) of an operation includes the operation name and the data types of it arguments(parameters).
- The implementation (or method) of the operation is specified separately and can be changed without affecting the interface .
- User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed as **program-operation independence** .

Data independence

- The capacity to change the schema at one level of a database system without having to change the schema at the next higher level
- Objective for 3-level architecture
- Similar concept to Abstract data types (ADT) in programming languages
- Hide implementation details from the users, to allow users to concentrate on the general structure , rather than on low-level implementation details

Kinds of data independence

- Logical data independence
- Physical data independence

Logical data independence

- Capacity to change the conceptual or logical schema without having to change external schema or application programs
- Changes to the conceptual schema includes:
    - Addition or removal of entities, attributes, or relationships
    - Expanding the database by addition of records or data item
    - Change of constraints

Physical data independence

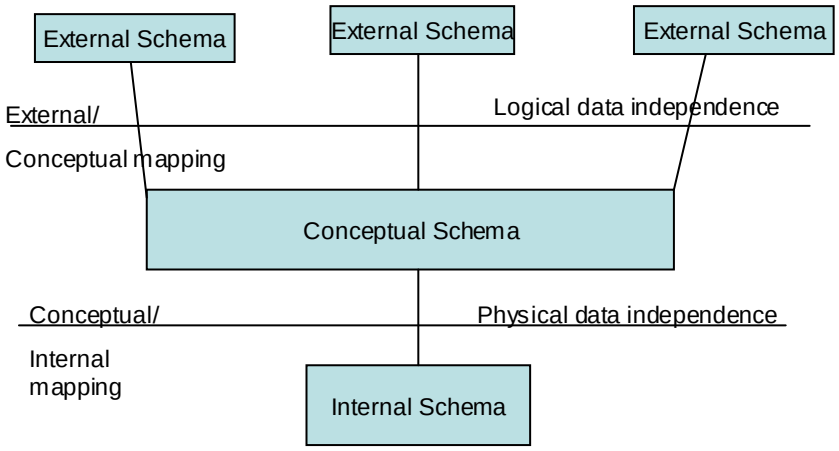
- Capacity to change the internal or physical schema without having to change the conceptual schema.
- Changes to the internal schema may be needed because some physical files had to be reorganized

e.g. by creating additional access structures to improve performance of retrieval or update

- Changes to the internal schema includes:
    - Different file organizations or storage structures, using different storage devices ,
    - Modifying indexes, or hashing algorithms

Fig. for data independence

- Fig.



Database languages :

- Data definition language(DDL)
- Storage definition language(SDL)
- View definition language(VDL)
- Data manipulation language(DML)
    - **DDL**: is a language which allows the DBA or user to describe and name the entities, attributes, and relationships required for the application , together with any associated integrity and security constraints.
    - **SDL**: is used to specify the internal(physical) schema .
    - **VDL**: is used to specify the user views and their mappings to the conceptual schema.
    - **DML**: It is a language that provides a set of operations on the data in the database. The operations include:
        - Insertion of new data in the database
        - Modification of data stored in the database
        - Retrieval of data contained in the database
        - Deletion of data from the database
- There are two main types of DMLs
    - **A high level or non procedural DML:** It is a language that allows the user to state what data is needed rather than how it is to be retrieved. High level DMLs, such as SQL , can specify and retrieve many records in a single DML statement; therefore , they are called **set-at-a-time or set-oriented DMLs.**
    - **A low-level or procedural DML:** It is a language that allows the user to tell the system what data is needed and exactly how to retrieve the data**.** This type of DML typically retrieves individual records or objects from the database and processes each separately. Therefore , it needs to use programming language constructs such as looping, to retrieve and process each record from a set of records. Low-level DMLs are also called **record-at-a-time DMLs.**
- **Query:** A query is a statement requesting the retrieval of information. A query in a high level DML often specifies which data to retrieve rather than how to retrieve it; therefore such languages are called **declarative**.

- **Query language:** When a high level DML used in a standalone interactive manner then we call it as a query language. Query language is a part of DML that involves data retrieval.
- **Host language:** Whenever DML commands , whether high level or low level , are embedded in a general-purpose programming language, that language is called the host language and the DML is called **data sublanguage.**
- **Data sublanguage:** These are the languages which do not include constructs for all computing needs such as conditional or iterative statements, which are provided by the high-level programming languages. A data sublanguage consists of two parts: DDL and DML.

Database interfaces: User-friendly interfaces provided by a DBMS may include the following:

- Menu-based interfaces for web clients or browsing
- Form-based interfaces
- Graphical user interfaces
- Natural language interfaces
- Speech input and output
- Interfaces for parametric users
- Interfaces for the DBA

ENTITY RELATION(E-R) MODEL

- Main features of ER model
- Role of E-R model in database
- Components of an E-R model

Main features of ER model

- A high level conceptual data model
- Allows us to describe the data involved in a real-world enterprise in terms of objects and their relationships
- Widely used to develop an initial design of a database
- Provides a set of useful concepts that make it convenient for a developer to move from a basic set of information to a detailed and precise description of information that can be easily implemented in a database system
- Describes data as a collection of entities, relationships and attributes

Role of E-R model in database

- ER model maps well to the relationship model.
- The constructs used in the ER model can be easily be transformed to relational tables
- Simple and easy to understand with a minimum of training
- The model can be used by the database designer to communicate the design to the end user
- The model can be used as a design plan by the database developer to implement a data model in specific database management software

Components of an E-R model
(Basic Constructs of E-R modeling)

- Entity
- Attribute
- Relationship
- Link

Entity Relationship Diagram (ERD)

- Used to visually represent data objects
- Most popular conceptual model
- The conceptual schema associated with ER model is called E-R diagram

Entities

- An entity is an object of concern used to represent the things in the real world
- e.g. car, table, book etc.
- An entity need not be a physical entity, it can also represent a concept in real world e.g. 'STUDENT' entity has instances of 'Ramesh' and 'Mohan'
- Bank Account is a conceptual entity

Entity type or Intension or Schema

- A collection (or set) of entities that have the same attributes
- e.g. EMPLOYEE and COMPANY are two entity types

Entity set (Extension)

- The collection of all entities of a particular entity type in the database at any point in time e.g.{ e1,e2, …,en} & {c1,c2,c3,…,cn}

where e={e1,e2,e3,….} is the entity set

and e1=(John,55,80k)
e2=(Fred,40,30k)
e3=(Judy,25,20k), …

are entity set for EMPLOYEE (Name, Age, Salary)

Entity occurrence (Instance)

- A uniquely identifiable object of an entity type
- A row in the relational table

Entity identifier key attributes

- An attribute whose values are distinct for each individual entity in the collection

Classification of entity types

- Strong entity type
- Weak entity type

Strong (or independent or owner or parent or dominant or regular)entity type

- An entity type that is not existence-dependent on some other entity type is called a strong entity typeThis is the entity type that must have one key attribute of it's own

Weak(dependent or child or subordinate) entity type

- It is an entity that is existence-dependent on some other entity type.
- The entity type which do not have any key attributes of their own

Attributes

- An attribute is a property used to describe the specific feature of the entity
- To describe an entity entirely, a set of attributes are used
- Example: A 'student 'entity may be described by the student's attributes like name , age , address , course, etc.

Domains (or Attribute domain or value sets of attributes) (V)

- A set of allowable values for one or more attributes
- Example: For a person entity person_id has a specific domain, integer values from 1 to 100(say)

Types of attributes

- Simple
- Composite
- Single valued
- Multi valued
- Derived
- Stored
- Not null/Null values
- Complex
- Key attributes

Simple (or atomic) attribute

- The attribute that can't be further divided into smaller parts and represents the basic meaning
- Example: The 'first name', 'last name', 'age' attributes of a person entity represent a simple attribute

Composite attribute

- Attribute that can be further divided into smaller units and each individual unit contains a specific meaning and composed of multiple components, each with an independent existence.
- Example: The 'Name' attribute of an 'employee' entity can be sub-divided into 'first name', 'last name' and 'middle name' .

Single valued attribute

- Any attribute that has single value for a particular entity
- Example: Age is a single-valued attribute of a person

Multi- valued attribute

- It is an attribute that can have a set of values for an entity
- Phone number ma lay be land phone as well as mobile phone.
- It may have some lower range and higher range of values

Derived attribute

- It is an attribute whose value is derived from value of some other attribute which is stored in the database.
- Example: (1)The 'age' attribute of a person is known as derived attribute because age can be derived from the attrute 'Birth_date' .
- (2)The attribute'year of experience' of an employee can be derived from the attribute 'date of joining(DOJ)'

Stored attribute

- An attribute is said to be stored attribute if it is stored permanently in the database
- The attributes 'Birth_Date' and 'Date_Of_Joining' are known as stored attributes ,which refers to the examples given in the previous slide

Not null/Null attribute values

- Any attribute whose value is unpredictable/unknown/not applicable/missing is known as null value.
- Any attribute whose value exist is called not null value

Complex attribute

- It is the nesting of composite and multi valued attributes
- We can represent arbitrary nesting by grouping components of a composite attribute between parentheses () and separating the components with commas, and by displaying multi valued attributes between braces { }

Example of Complex attribute

- If a person can have more than one residence and each residence can have a singe address and multiple phones, an attribute address_phone for a person can be specified as shown in figure below
- {Address_phone({Phone(Area_code,Phone_number)},Address(Street_address(Number, Street,Apartment_number),City,State,Zip

Key attributes

- A key is an attribute or set of attributes whose values are unique (distinct or not repeated) for an entity type.

Different types of Keys

- Composite key
- Primary key

- Super key
- Candidate key
- Alternate (or secondary) key
- Foreign key
- Artificial key

Composite key

- When a key contains more than one attribute, then we call it as a composite key

Primary key (PK)

- A key is an attribute (or set of attributes) whose value (or values) is (or are) unique and not null .

  So PK=UNIQUE and NOT NULL

Foreign Key (FK)

- A foreign key is an attribute (or set of attributes) whose value (or values) matches with the value(or values) of primary key and the values of the foreign key may be repeated and may be null.
- So, FK=PK=(may be repeated values of PK)=(may be NULL)

Super Key (SK)

- A super key is a set of one or more attributes whose values collectively , allows us to identify uniquely a tuple(or an entity) within a relation (or in the entity set)
- Any superset of key (K) is the super key (SK)
- Key is always called as a super key ,which we call as minimal super key, because any set is superset of it self.
- Example: ConsiderSTUDENT(ROLLNO,NAME,CLASS,MARKS) relation/table/entity
- Here {ROLLNO} is unique and is the key as well as super key (minimal super key) of the 'STUDENT' entity
- So all the supersets of {ROLLNO} are {ROLLNO}, {ROLLNO,NAME} , {ROLLNO,NAME,CLASS}, {ROLLNO,NAME,CLASS,MARKS} are all the super keys of 'STUDENT' table/relation/entity
- If K is a super key then it's all supersets are called super keys
- A super key may contain extraneous attributes
- A super key can have redundant attributes
- Every relation/table/entity has at least one default super key-the set of all it's attributes

Candidate Key (CK)

- A candidate key is a super key such that no proper subset is a super key within the relation/table/entity i.e.Combined values of candidate key (CK) is always unique where as all the proper subsets of CK are not unique
- A candidate key, CK for a relation R has two properties:
  - Uniqueness: In each tuple of R, the value of CK uniquely identify that tuple.
  - Irreducibility: No proper subset of CK has the uniqueness property.
- Minimal super keys are called keys/candidate keys/primary keys
- Every relation does have at least one candidate key
- There can be any number of candidate keys in a table
- If a relation has more than one key, then each of the keys is called a candidate key.
- A super key has the uniqueness property but not necessarily the irreducibility property
- The superset of candidate keys are called super keys
- A primary key is the candidate key that is selected to identify tuples uniquely within the relation
- It is better to choose a primary key with a single attribute or small number of attributes

Alternate (or secondary) Key

- The candidate keys that are not selected to be the primary key are called alternate keys
- AK=CK-PK
- A secondary key is an attribute (s) that is used strictly for data retrieval
- If in a table there are n number of candidate keys then one of them (minimal) is taken as primary key and others(n-1 numbers) are known as alternate keys
- Consider a relation PERSON(person_id,name)
- Here peron_id is the primary key and name will be alternate key (Assume name is unique)

Artificial Keys

- An artificial key is the one that has no meaning to the business or organization
- Artificial keys are permitted when
  - No attribute has all the primary key properties, or
  - The primary key is large and complex

- The table enrollment (student, class,…..) from the business point of view has (student, class) combination as primary key. But this primary key is large and complex, so DBA decides to add row_id column as primary key

Properties of Primary Key:

- Stable: The value of a primary key must not change or should not become null throughout the life of an entity.
- Minimal: The primary key should be composed of the minimum number of fields that ensure the occurrences are unique.
- Definitive: A value must exit for every record at creation time. Because an entity occurrence can not be substantiated unless the primary key value exists.
- Accessible: Any one who wants to create, read or delete a record must be able to see the primary key value.

Properties of Candidate key:

- A candidate key must be unique.
- A candidate key's value must exist.. It can't be null. (This is also called entity integrity rule)
- A candidate key is a minimal set of attributes.
- The value of candidate key must be stable. Its value can't change outside the control of the system.

Relationships

A relationship can be defined as:

- A connection or set of associations, or
- A rule for communication among entities

Relationship type

- A relationship type R among n entity types E1,E2,E3,…,En is defined as a set of meaningful associations among entity types

Relationship set

- A relationship set is a set of relationships of the same type. of relationship opts forms a relationship set called relationship type
- Collection of all the instance of relationship 'opts' forms a relationship set called relationship type

Relationship occurrence

- It is a uniquely identifiable association ,which includes one occurrence from each participating entity type, is called e relationship occurrence

Degree of a relationship type

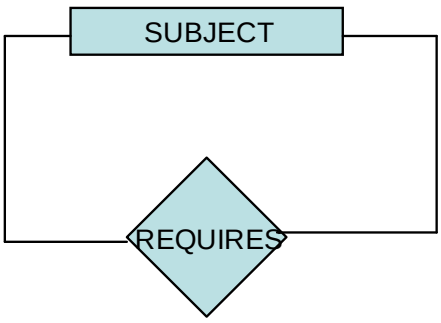
- It is the number of participating entity types in a relationship

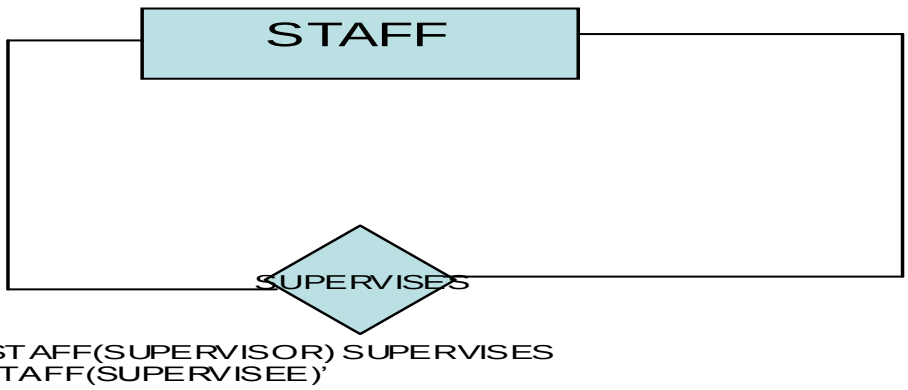
Types of relationships

- Unary
- Binary
- Ternary
- Quaternary
- n-ary

Unary/Recursive relationship

- It is an association with a single entity
- Example:-'statistics' requires 'mathematics' and 'English drama' requires 'English'





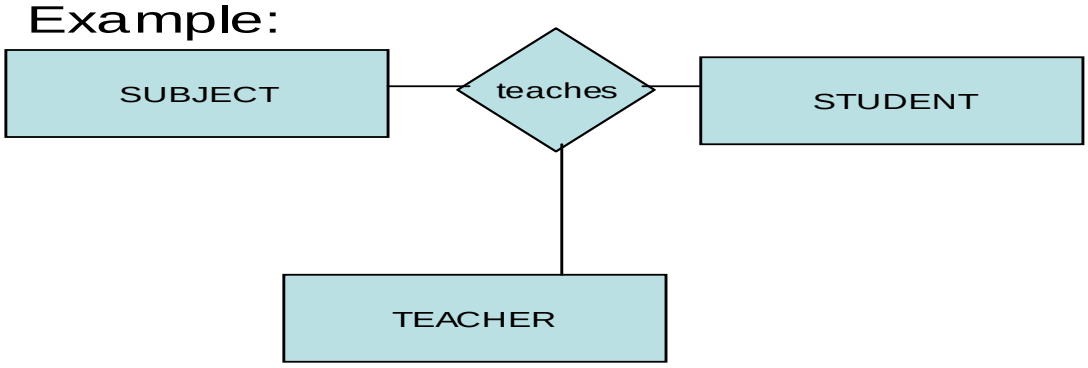
'STAFF(SUPERVISOR) SUPERVISES STAFF(SUPERVISEE)'

Binary relationship

- It is the relationship between two entities
- Example:

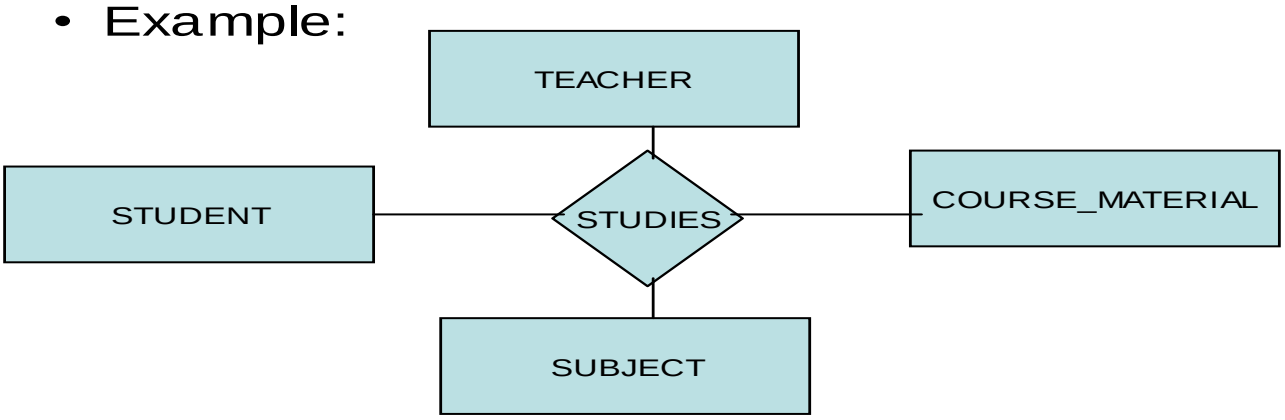
Ternary relationship

- It is a relation among three entities
- Example:



Quaternary relationship

- It is a relationship among four entities
- Example:



Role names

- The role name signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.

- Role name indicates the purpose that each participating entity type plays in a relationship
- Role names are technically necessary in relationship types where all the participating entity type name can be used as the roll name
- Example: In the 'WORKS_FOR' relationship type ,'EMPLOYEE' entity plays the role of employee or worker and ' DEPARTMENT' entity plays the role of department or employer

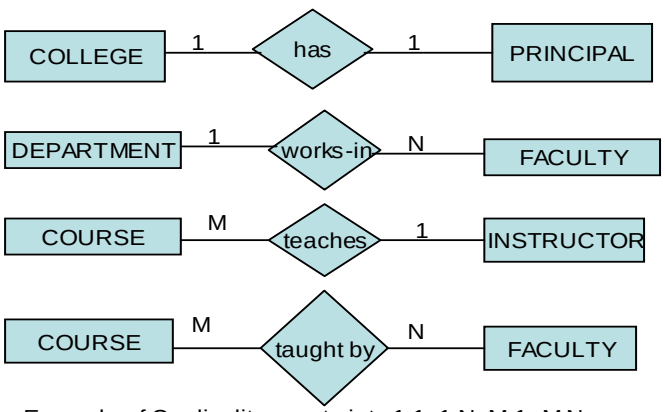
Constraints on relationship types

- Connectivity
- Cardinality(for binary relationship) or cardinality constraints
- Participation constraint
- Structural constraint

Connectivity

- Connectivity of a relationship describes the mapping of associated entity instances in the relationship
- Cardinality(for binary relationship) or cardinality constraints or Basic types of connectivity for relations
- Cardinality is the maximum number of possible relationship occurrences(or instances) for an entity participating in a given relationship type
- Cardinality constraint is classified into 4 categories:
  - One-to-one (1:1)
  - One-to-many(1:N) or (1:*)
  - Many-to-one(M:1) or (*:1)
  - Many-to-many (M:N) or (*:*)

- One-to-one(1:1): An entity in A is associated with at most one entity in entity B, and an entity in B is associated with at most one entity in A
- One-to-many(1:N): An entity in A is associated with any number of entities in B and an entity in B is associated with at most one entity in A.
- Many-to-one(M:1): An entity in A is associated with at most one entity in B and. an entity in B is associated with any number in A.
- Many-to-many(M:N): Entities in A and B are associated with any number of entities from each other.



Example of Cardinality constraints:1:1, 1:N, M:1, M:N

Participation constraint
(Minimum cardinality constraint)

- Participation :-determines whether all or only some entity occurrences participate in a relationship
- Participation constraints specify whether the existence of an entity depends on its being related to another entity via the relationship type.
- It is the minimum number of relationship instances that each entity can participate in and is some times called the minimum cardinality constraint

- Participation constraints are of two types:
  - Total (mandatory) participation constraint (Existence dependency)
  - Partial (Optional) participation constraint

Total (mandatory) participation constraint (Existence dependency)

- When all the entities from an entity set participate in a relationship type, then it is called total participation
- Example: The participation of the entity set ‘student’ in the relationship set must ‘opts’ is said to be total because every student ‘enrolled ‘must opt for a ‘course’

Partial (Optional) participation constraint

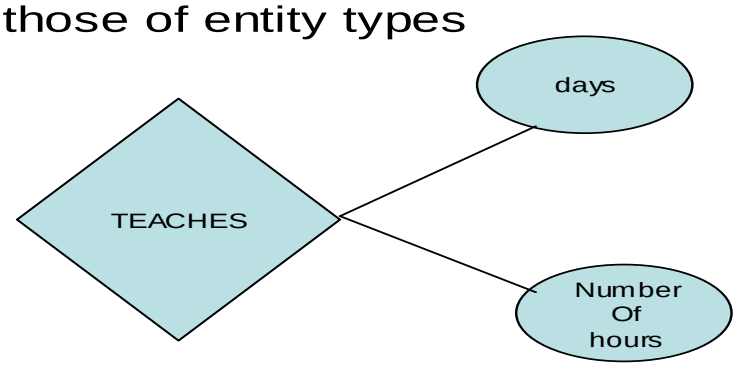
- When it is not necessary for all the entities from an entity set to participate in a relationship type, it is called partial participation
- Example: The participation of the entity set ‘student’ in relationship ‘represents’ is partial, since not every student in a class is a class representative

Structural constraints of a relationship type

- The cardinality ratio and participation constraints taken together is called structural constraints

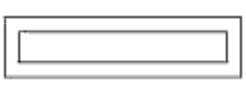
Attributes of relationship type

- Relationship types can have attributes, similar to those of entity types
- Example:



Symbols used in ER diagram

ENTITY TYPE



WEAK ENTITY TYPE

RELATIONSHIP TYPE

ATTRIBUTE

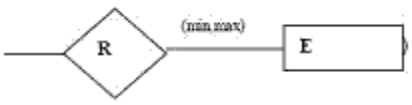
KEY ATTRIBUTE

MULTIVALUED ATTRIBUTE

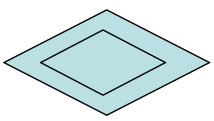
DERIVED ATTRIBUTE

TOTAL PARTICIPATION OF E2 IN R

Cardinality Ratio 1:N FOR E1:E2 IN R



Structural Constraint(Min,Max) On Participation Of E In R



Identifying relationship

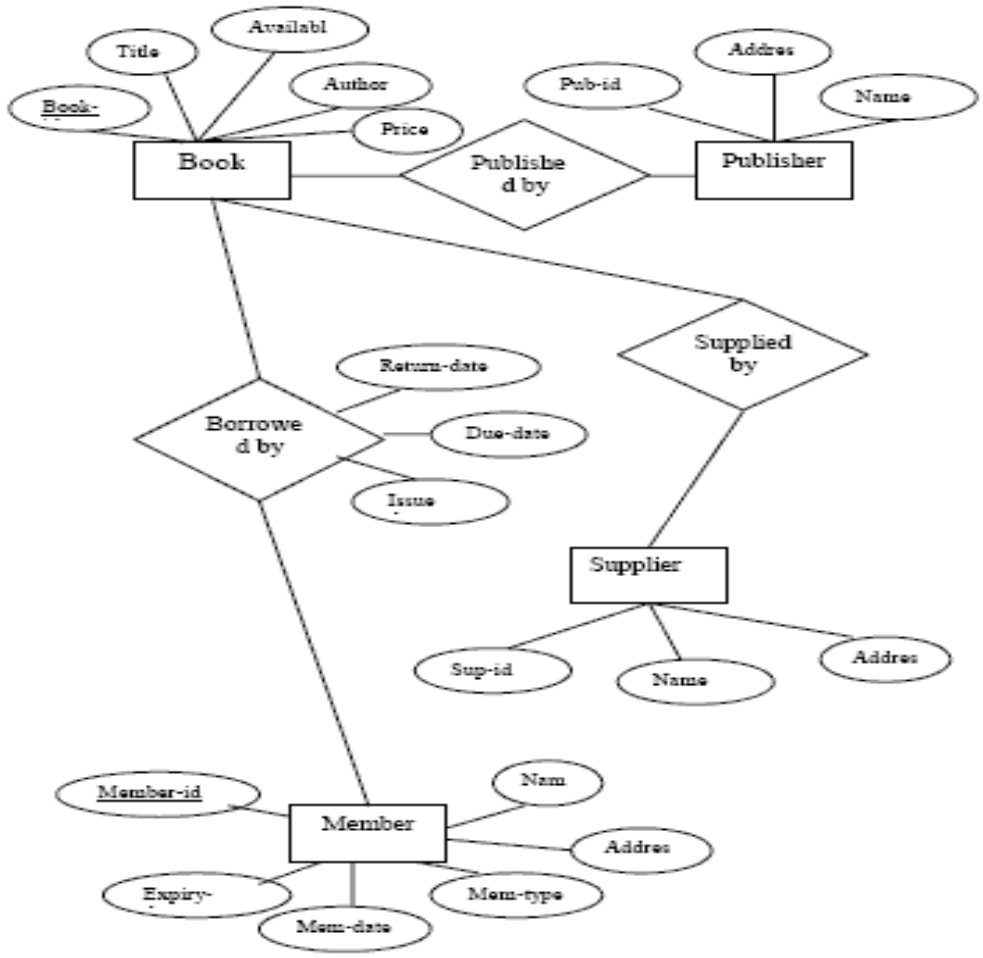
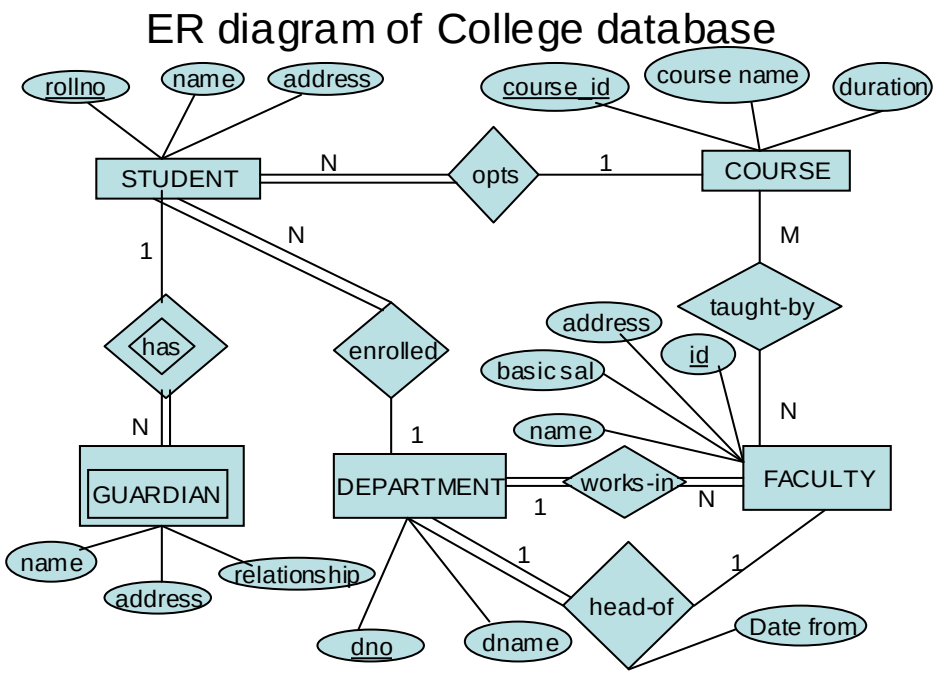


Fig. E-R Diagram of Library Management System.



Enhanced Entity Relationship (EER) Model

- It is the ER model with additional semantic concepts (or enhanced features)
- The basic features of ER diagrams are sufficient to design many database situations but with more complex relations and advanced database applications, it is required to move to enhanced features of ER models
- The enhanced features are :
  - Generalization
  - Specialization
  - Aggregation
  - Categorization
  - Composition

Class

- A class is a set of objects (grouping of objects) that have identical properties, common behavior , and shared relationship
- Examples of class are the following:
- Fruit is a class whose objects are mango, banana, apple, and orange.
- Employee is a class whose object are principal, teacher, clerk and peon.
- Furniture is a class whose objects are chair, table and cup board
- Account is a class whose objects are saving account and current account

Object

- In the ordinary sense, an object is something which is capable of being seen, touched or sensed
- In programming sense, objects are the basic run time entities (or instances of class)

Inheritance

- Deriving a new class (child class or sub class or sub type class or specialization class) from an old class (existing class or parent class or super class or super type class or generalization class) is called an inheritance.
- A sub class must have all the properties of the super class and other properties as well.

Super class/subclass relationship(IS-A or IS-AN relationship)

- It is the relationship between a super class and any of its subclass, which is often called as an IS-A or IS-AN relationship.
- Example: A SECRETARY is an EMPLOYEE, A TECHNICIAN is an EMPLOYEE .

Attribute inheritance:

- Attribute inheritance is a concept/property in which a subclass(specialization class) inherits attributes (of higher level and lower level entity sets) and relationships of the class(super/generalization class) from which it is derived.
- The attributes of higher level entity sets(super class) are inherited by the lower level entity sets(sub class).

Type hierarchy:

- An entity(super class) and its subclasses and their subclasses, and so on, is called a type hierarchy.
- Example: 'Manager' is a specialization of 'Staff', 'Staff' is a generalization of 'Manager', 'Manager' IS-A member of 'STAFF'

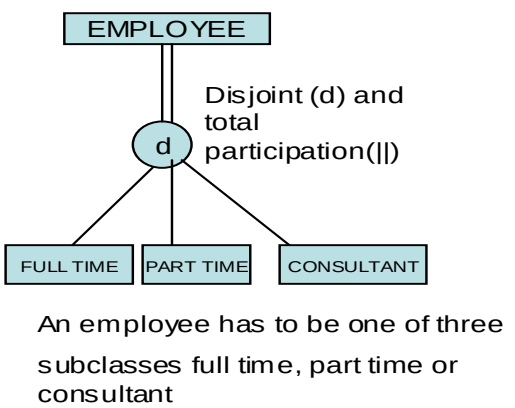
Shared subclass: A subclass with more than one super class is called a shared subclass.
Multiple inheritance: When the attributes of the super classes are inherited by the shared subclass, which may also have its own additional attributes, then we call it as multiple inheritance.

Generalization/Specialization:

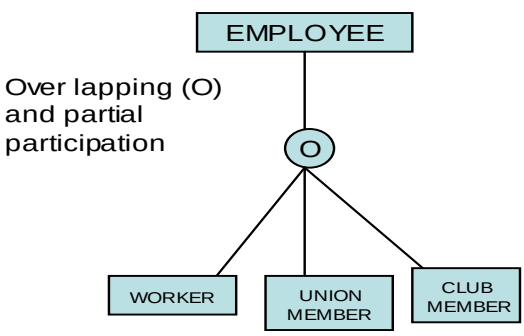
- It is a technique where in the attributes and behaviors that are common to several types of object classes(subclasses) are grouped into their own class(super type/super class).

Constraints on Specialization/Generalization:

- Participation constraint
- Disjoint constraint
    - Participation constraint: It determines whether every member in the super class must participate as a member of a subclass.
    - Disjoint constraint: It describes the relationship between members of the subclasses and indicates whether it is possible for a member of a super class to be a member of one, or more than one, subclass.
    - Examples:



An employee has to be one of three subclasses full time, part time or consultant



An employee can be a worker as well as union member as well as a club member

Enhanced features of EER model

- Generalization
- Specialization
- Aggregation
- Composition
- Categorization

Generalization: Generalization is the result of taking the union of several lower level entity sets (sub class) to produce higher level entity sets(super class).

Or

It is the process of minimizing the differences between entities by identifying their common characteristics

Or

It is the process of identifying some common characteristics of a collection of entity sets and creating a new entity set that contains entities processing these common characteristics.

Specialization: Specialization is the result of taking subsets of a higher level entity set to form lower level entity sets.

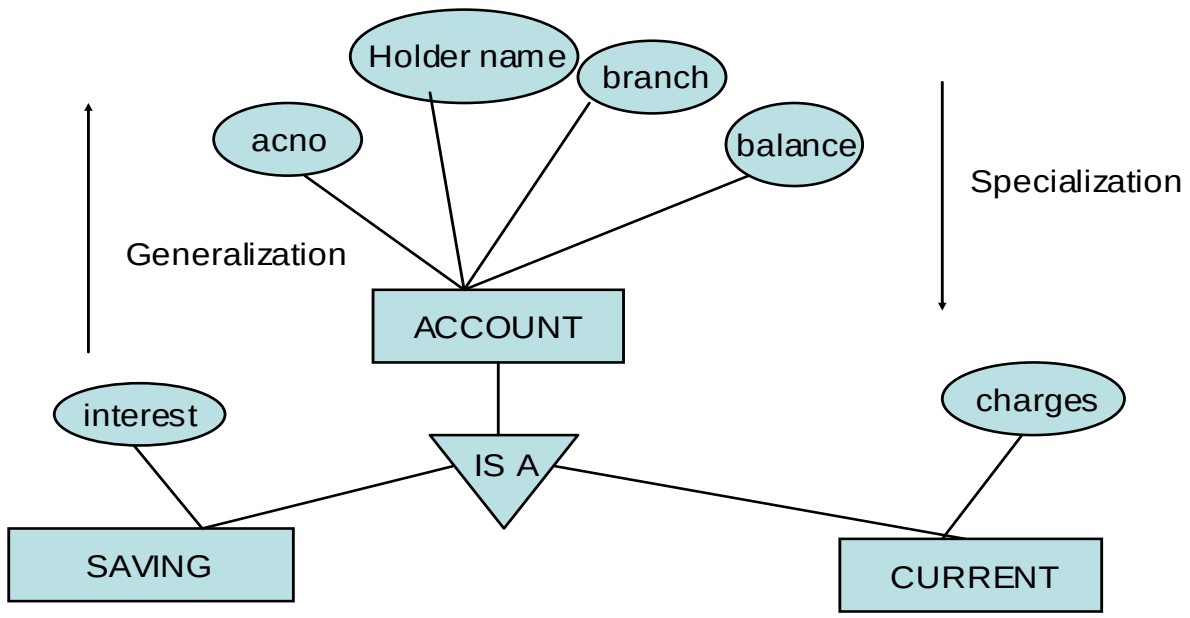
Or

It is the process of maximizing the differences between members of entity by identifying their distinguishing characteristics.

Or

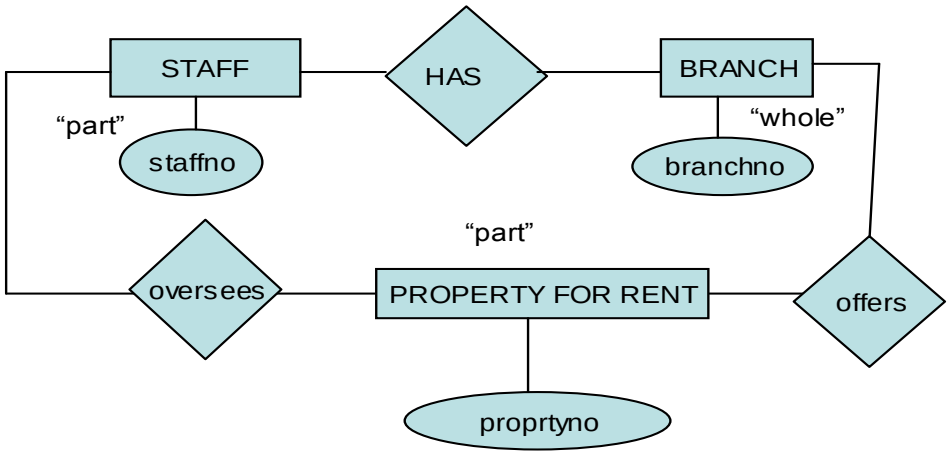
It is the process of identifying subsets(subclass) of an entity set(the super class) that share some distinguishing characteristics.

Example is in the next page:

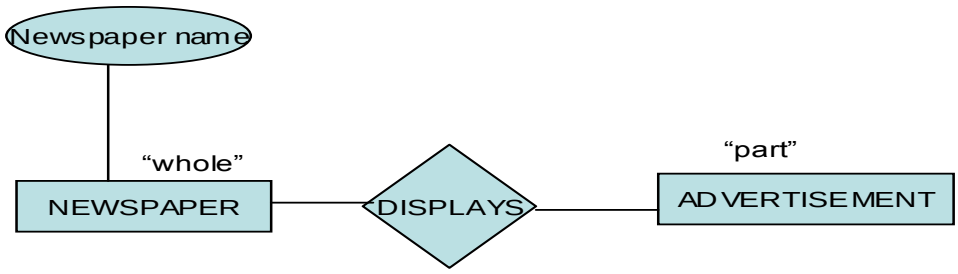


Aggregation: Aggregation represents a 'has-a' or 'is-part-of' relationship between entity types, where one represents the 'whole' and the other the 'part'.

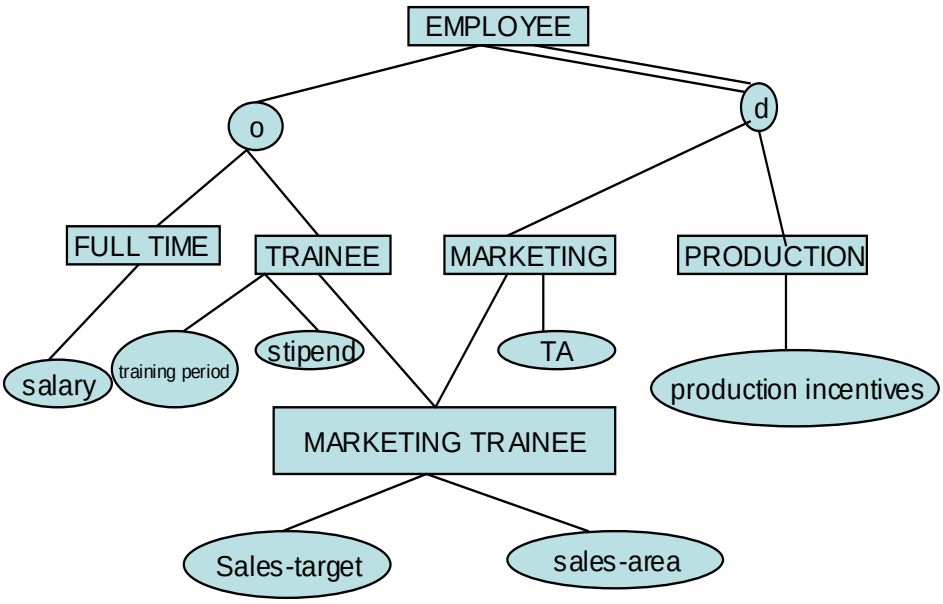
Example:



Composition: Composition is a specific form of aggregation that represents an association between entities, where there is a strong ownership and coincidental lifetime between the 'whole' and the 'part'.
Example:



Categorization: The modeling of a single subclass with a relationship that involves more than one distinct super class is called categorization.
Example:



'MARKETING TRAINEE' subclass has more than one super class 'TRAINEE' and 'MARKETING', called as a category and the process of defining a category is called categorization. "O" stands for overlapping constraints where as "d" stands for disjoint constraints and double line before "d" represents the total participation where as single line before "O" represents the partial participation.

The relational data model

- A relational model is a simple model in which database is represented as a collection of "Relations", where each relation is represented by a 2D(dimensional) table
- It was first proposed by Dr. E.F.Codd (the researcher of IBM in the year 1970 in his seminar paper)
- The first commercial implementations of the relational model became available in the early 1980s, such as the Oracle DBMS and the SQL/DS system on the MVS operating system
- Current popular relational DBMSs (RDBMSs) are:
  - DB2
  - Informaix Dynamic server(from IBM)
  - Oracle and Rdb (from oracle)
  - SQL server and Access(from Microsoft)

Objectives of the relational model

- To allow a high degree of data independence
- To provide substantial grounds for dealing with data semantics, consistency, and redundancy problems
- To enable the expansion of set-oriented data manipulation languages
- The development of a structured query language (SQL)
- The production of various commercial RDBMS products during the late 1970s and the late 1980s

Relational database management system (RDBMS)

- Dr. E.F. Codd outlined the principles of the relational model, which formed the basis for the evolution of the RDBMS and defined a relational database management system (RDBMS) as a
  - Collection of tables related to each other through common values

Or

- A relational model database is defined as a database that allows you to group its data items into one or more values independent tables that can be related to one another by using fields (columns or attributes) common to each related table.
- Examples of RDBMS are:
  - DB2,ORACLE,Sybase,MS-SQL SERVER,MS_ACCESS,INGRESS

Codd's 12 rules of RDBMS

- Dr. E.F.Codd was the creator of the relational data model
- It was published as a two-part article in computer world(Codd, 1985).
- It contains a list of 12 rules that determine whether a DBMS is relational and to what extent it is relational
- These rules are a very useful yardstick for evaluating a relational system
- In the article Codd mentioned that according to these rules , there is no database yet that is fully a relational system
- He says that the rules 6,9,10,11, and 12 are difficult to satisfy
- Codd's 12 rules are:
  - Information rule
  - Guaranteed access rule
  - Systematic treatment of null values
  - Self-describing database
  - Comprehensive data sub language
  - View updating rule
  - High-level insert, update and delete
  - Physical data independence
  - Logical data independence
  - Integrity independence
  - Distribution independence
  - Non subversion rule
- ***Rule-1:Information rule***
  - This rule states that all data is represented as tables.
  - Data can be viewed in no other way
- ***Rule-2:Guaranteed access rule***
  - Each data item must be accessible by providing table name+ primary key of the row+ column name
- ***Rule-3:Systematic treatment of null values***
  - Handle missing and /or non-applicable data, that is , nulls, in a consistent manner
  - Nulls should have no values and should simply be missing data
  - Treating null as zero for missing numeric data or as a blank for missing character data violates this rule
  - Vendors provide the ability to use default values for missing data, if so desired
- ***Rule-4:Self-describing database***
  - In addition to storing user data, a relational database must contain data about itself
  - There re two types of tables in RDBMS:
  - User tables, which contain the data required by users, and
  - System tables, which contains the data about the database structure
  - The data that describes the database structure is called meta data
  - The collection of system tables is called as system catalog or data dictionary
  - A data dictionary holds the following in formations about each data element in the database: name, type, range of values, source, access authorization

- Storing any part of the data dictionary in operating system files would violate this rule

- ***Rule-5:Comprehensive data sub language***
  - The language must support relational operators and set operations with regard to the following:
    - Data definition
    - Data manipulation
    - Integrity definition
    - Transaction control
    - Data control
    - Accessing data files (files that contain the actual data), through a utility other than an SQL interface, violates this rule
- ***Rule-6:View updating rule***
  - This rules states that views should allow updates in the underlying tables and vice versa
  - Views are virtual tables, which unlike a table contains no data, just statements that return data in the form of tables
  - But SQL supports only updates of single tables at a time , therefore if , we join three table to create a view , and try to update the view , then DBMS would fail to translate these updates to underlying tables, thereby violating this rule
  - Also , a view not including the column that uniquely identifies each record in a table can not be updated, thus violating the rule
- ***Rule-7:High-level insert, update and delete***
  - This rule states that in a relational database , the query language(SQL) must be capable of performing manipulations(such as, inserting , updating, or deleting data) on sets of rows in a table
  - A database that supports only row-at-a-time manipulation can not be considered as relational
- ***Rule-8:Physical data independence***
  - This rule states that in a relational database, any changes made in the way data is physically stored(that is, data stored in file systems specific to the machine operating system) must not affect applications that access data
  - If a file supporting a table was moved from one disk to another, or renamed then this should have no impact on the application
- ***Rule-9: Logical data independence***
  - This rule states that logical changes in views/tables such as adding or deleting columns or modifying field lengths do not necessitate modification in the application programs or ad hoc request formats
- ***Rule-10: Integrity independence***
  - Integrity constraints or data integrity rules that should apply to relational database are:
  - Entity integrity –the primary key column can not have missing values
  - Referential integrity-for every foreign key column value there must exist a matching primary key column value
  - This rule states that
    - Integrity constraints specific to a particular relational database must be defined in the relational data sublanguage and storable in the catalog , not
- ***Rule-11: Distribution independence***
  - In a relational database , data can be stored centrally(on a single system) or distributed(across a network of systems)

- The data in a centralized database should remain logically unaffected if they are distributed across systems
- For example , a user should be able to retrieve data from two tables distributed across two terminals, the same way, as they would retrieve them in the same terminal

- ***Rule-12:Non subversion rule***
  - This rule states that there should be no other access path to be obeyed the database, other than SQL.
  - Any other access language may bypass (or subvert) security or integrity rules, which otherwise would be obeyed by the regular data access language

Major components of a relational data model

- Relational data structure (data structure)
- Relational data integrity(relational integrity) (data integrity) (Relational data model constraints)
- Relational data manipulation (data manipulation)
- ***Relational data structure***
  - It is the set of relations (tables) and set of domains that defines the way the data can be represented
- ***Relational data integrity***
  - It is the integrity rules that define the procedure to protect the data
- ***Relational data manipulation***
  - It is the operation that can be performed on data

Relational data structure

- Relation
- Attribute
- Domain
- Tuple
- Extension of relation
- Intension of a relation
- Degree
- Cardinality

- ***Relation (R)***
  - A relation is a table (entity) with columns (or attributes or field) and rows (records or tuples)
  - A relation or database relation consists of
  - Relational schema (Intension)
  - Relational instance (Extension)
  - A relational schema is a named relation defined by a set of attributes and domain pairs
  - A relational instance is a collection of tuples for a given relational schema at a specific point of time

- ***Properties (or characteristics) of relations***
  - The relation has a name that is distinct from all other names in the relational schema
  - Each cell of the relation contains exactly one atomic(single) value
  - Each attribute has a distinct name
  - The values of an attribute are all from the same domain
  - Each tuple is distinct ; there are no duplicate tuples
  - The order of the attributes has no significance

- The order of the tuple has no significance
- The relation schema can be interpreted as a declaration or type of assertion
- Each tuple in the relation can be interpreted as a fact or a particular instance of the assertion
- Some relations may represent facts about entities, where as other relations may represent facts about relationships

- ***Attribute (A) or Column or field***
  - An attribute is a named column of a relation
- ***Domain (D)***
  - A domain is a set of allowable values for one or more attributes
  - A data type or format is also specified for each domain

    or
  - A domain is a set of atomic values.
  - By atomic we mean that each value in the domain is indivisible as far as the relational model is concerned
- ***Tuple (t)(Row or record)***
  - A tuple is a row of a relation
  - Extension of a relation is the set of tuples appearing at any given instant of time. It varies with time . It changes as tuples are created, destroyed, and updated
  - Intension of a relation is the structure of the relation together with a specification of the domains and any other restrictions(integrity constraints) on possible values. It is the permanent part of the relation and independent of time
- ***Degree (arity)***
  - The degree of a relation is the number of attributes it contains
  - It is the property of intension of the relation
  - Kinds of degree
  - 1 (Unary): Relation with only one attribute (degree=1)
  - 2(Binary):Relation with only two attributes (degree=2)
  - 3(Ternary):Relation with only three attributes (degree=3)
  - n>=4(n-ary):Relation with only 'n' attributes (degree>=4)
- ***Cardinality***
  - The cardinality of a relation is the number of tuples it contains
  - It is the property of the extension of the relation

Relational data integrity or relational model constraints: Constraints or restrictions on databases
can generally be divided into following four main categories :

- Inherent model based constraints
- Schema based constraints (Constraints on relation)
- Application based constraints
- Enterprise constraints

- ***Inherent model based constraints***
  - Constraints that are inherent in the data model are called the inherent model based constraints
  - The constraint that a relation can not have duplicate tuples is an inherent constraint
- ***Schema based constraints (Constraints on relation)***
  - Constraints that can be expressed in the schema of the relational model via DDL is known as schema based constraint
  - The schema based constraints are
    - Domain constraints

- Key constraints
- Constraints on nulls
- Integrity constraints
  - Entity integrity constraint
  - Referential integrity constraint

- ***Application based constraints***

Constraints that can not be directly expressed in the schema of the data model , and hence must be expressed and enforced by the application programs, are called as application-based constraints

- ***Enterprise constraints***
  - Enterprise constraints are additional rules specified by the users or database administrators of a database
  - The enterprise constraints are
    - State(Static) constraints
    - Semantic integrity constraints (Tiggers and Assertions)
    - Data dependency constraints (FD & MVD)

Schema based constraints (Constraints on relation)

- The schema based constraints are
  - Domain constraints
  - Key constraints
  - Constraints on nulls
  - Integrity constraints

Domain constraints

- It specifies that each attribute in a relation must contain an atomic value only for corresponding domains of attributes
- The data types associated with commercial RDBMS include:
  - Standard numeric data types for integer
  - Real numbers
  - Characters
  - Fixed length strings and variable length strings

Key constraints

- It states that the key attribute value in each tuple must be unique i.e. no two tuples contain the same value for the key attribute
- This is because the value of the primary key is used to identify the tuples in the relation

Constraints on nulls

- Null represents a value for an attribute that is currently unknown or is not applicable for this tuple
- A constraint on attribute specifies whether NULL values are or are not permitted

NAME	AGE	JOB
Rajesh		Clerk
Raja	23	
Amit	43	Sales

Integrity constraints

- It provide a means of ensuring that changes made to the database by authorized users do not result in a loss of data consistency
- There are two types of integrity constraints:
  - Entity integrity constraints

- Referential integrity constraints

Entity integrity constraints

- ❖ It states that in a base relation no attributes of a primary can be null
- ❖ 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

Referential integrity constraints

- ❖ It states that if a foreign key exists in a relation either the foreign key value must match a candidate (or primary) key value of some tuple in its home relation or the foreign key value must be wholly null

Enterprise constraints

- ❖ The enterprise constraints are
  - State(Static) constraints
  - Semantic integrity constraints (Triggers and Assertions)
  - Data dependency constraints (FD & MVD)
  - Transition (Dynamic) constraints

State(Static) constraints

- ❖ It define the constraints that a valid state of the database must satisfy

Semantic integrity constraints

- ❖ It can be specified and enforced within the application programs that update the database, or by using a general purpose constraint specification language(such as SQL-99 uses triggers and assertions for this purpose)

Data dependency constraints

- ❖ It includes functional dependencies (FD) and multi valued dependencies (MVD), which are used mainly for testing the "goodness" of the design of a relational database and are utilized in a process called normalization

Transition (Dynamic) constraints

- ❖ It can be defined to deal with state changes in the database
- ❖ EX. "The salary of an employee can only increase". Such constraints are typically enforced by the application programs or specified using active rules and triggers

Relational data manipulation

- ❖ The manipulative part of the relational model consists of a set of operators known collectively as the relational algebra together with relational calculus

Relational Algebra

- ❖ It is a procedural language that can be used to tell the DBMS how to build a new relation from one or more relations in the database
- ❖ Relational algebra is a set of basic operations used to manipulate the data in a relational data model
- ❖ PL/SQL is a procedural language

Relational Calculus

- ❖ It is a non-procedural language that can be used to formulate the definition of a relation in terms of one or more database relations
- ❖ SQL is a non-procedural language

Comparison of relational algebra and relational calculus

- ❖ The relational algebra and the relational calculus are two formal languages for the relational model.
- ❖ The relational algebra is a procedural language of solving the queries where as the relational calculus is a non procedural language of solving the queries.
- ❖ The solution to the database access problem using a relational algebra is obtained by stating what is required and what are the steps to obtain that information where as the

solution to the database access problem using a relational calculus is obtained simply by stating what is required and letting the system find the answer.

- The relational algebra is used as a vehicle for implementation of relational calculus where as relational calculus queries are converted into equivalent relational algebra format by using Codd's reduction algorithm and then it is implemented with the help of relational algebra operators.
- Relational algebra operators are used as a yardstick for measuring the expressive power of any given language, where as a language is said to be complete if it is at least powerful as the calculus i.e., if any relation definable by some expression of the language in question.

Relational algebra expression: A sequence of relational algebra operations forms a relational Algebra expression, whose result will also be a relation that represents the result of the database query or retrieval request.

Why the relational algebra is very important? Reasons

- It provides formal foundation for the relational model operations
- It is used as a basis for implementing and optimizing queries in relational database management systems (RDBMSs)
- Some of its concepts are incorporated into the SQL for RDBMSs

Relational Algebra operations

- Basic set operations (Binary operations for sets)
  - Union
  - Intersection
  - Set difference
  - Cartesian or Cross Product or Cross Join
- Relational operations
  - Unary relational operations
    - Select
    - Project
    - Rename
  - Binary relational operations
    - Cross join (Cartesian or Cross Product)
    - Join
    - Self join
    - Equi join
    - Non equijoin
    - Natural join
    - Semi join
    - Inner join
    - Outer join (left outer join , right outer join and Full outer join)
    - Division
- **Union Compatible**: Two relations R1(A1,A2,A3,…,An) and R2(B1,B2,B3,…,Bn) are said to be union compatible if
  - Degree(R1)=Degree(R2)=n i.e. number attributes of two relations R1 and R2 are same.
  - Domain(Ai)=Domain(Bi) for 1<=i<=n i.e. the data types of corresponding attributes of R1 and R2 are same.
- **Basic set operations in relational algebra(Binary operations for sets)**
  - Union

- Intersection
- Set difference
- Cartesian or Cross Product or Cross Join

- **Union:(U):**
- R1U R2={ t |t belongs to R1 or t belongs to R2 Or(t belongs to R1 and t belongs to R2)}
- If R1 and R2 are union compatible then R1UR2 is the union of two relations R1 and R2 which defines a relation that contains all the tuples of R1, or R2, or both R1 and R2, duplicate tuples being eliminated.
- **Properties of union**:
  - **Commutative**: R1UR2=R2UR1
  - **Associative**:R1U(R2UR3)=(R1UR2)UR3
- **Intersection:(∩):**
- R1∩R2={t| t belongs to R1 and t belongs to R2}
- If R1 and R2 are union compatible then R1∩R2 is the intersection of two relations R1 and R2 which defines a relation that contains set of all tuples that are in both R1 and R2.
- **Properties of intersection**:
  - **Commutative**: R1∩R2= R2∩R1
  - **Associative**:R1∩(R2∩R3)=(R1∩R2) ∩R3
- **Set difference(minus)(-)**
- R1-R2={t|t belongs to R1 but t doesn't belong to R2}
- If R1 and R2 are union compatible then R1-R2 is the set difference of two relations R1 and R2 that defines a relation consisting of the tuples that are in relation R1, but not in relation R2.
- **Properties of set difference:**
  - **Not commutative**: R1-R2≠R2-R1
  - **Not associative**:R1-(R2-R3) ≠(R1-R2)-R3
  - **R-(R-S)=R∩S**
- **Cartesian(or Cross) Product(or Cross join)(X):**
- R1XR2 is the Cartesian product of two relations R1 and R2 , which defines a relation that is the concatenation of every tuple of relation R1 with every tuple of the relation R2.
- **Properties of Cartesian product:**
  - **Commutative:** R1XR2=R2XR1
  - **Associative:** R1X(R2XR3)=(R1XR2)XR3
  - **Degree(R1XR2)=Degree(R1)+Degree(R2)**
  - **Cardinality(R1XR2)=Cardinality(R1).Cardinality(R2)**

    i.e.|R1XR2|=|R1||R2|
- **Example to illustrate union, intersection, set difference and Cartesian product:**

  Consider two relations R1 and R2

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y

A1	B1
A7	B7
A2	B2
A4	B4

- **Find R1UR2, R1∩R2, R1-R2 and R1XR2.**
- **R1UR2=R3**

A	B
A1	B1
A2	B2
A3	B3
A4	B4
A7	B7

- **R1∩R2=R4**

A	B
A1	B1
A2	B2
A7	B7

- **R1-R2=R5**

A	B
A3	B3

- **R1XR2=R6**

A	B	X	Y
A1	B1	A1	B1
A1	B1	A7	B7
A1	B1	A2	B2
A1	B1	A4	B4
A2	B2	A1	B1
A2	B2	A7	B7
A2	B2	A2	B2
A2	B2	A4	B4
A3	B3	A1	B1
A3	B3	A7	B7
A3	B3	A2	B2
A3	B3	A4	B4
A4	B4	A1	B1
A4	B4	A7	B7
A4	B4	A2	B2
A4	B4	A4	B4

- **Relational operations in relational algebra**
  - Unary relational operations: operate on single relations
    - Select
    - Project
    - Rename
- **Selection(**□□□□□The selection operation works on a single relation R that defines a relation that contains only those tuples of R that satisfy the specified condition or predicate and it is denoted by the symbol (sigma)

σ <selection condition> (R)

❖ The Boolean expression specified in <select condition> is made of a number of clauses of the

form : <attribute name> <comparison operator><constant value> or
<attribute name><comparison operator><attribute name>
<comparison operator> ={ <=,>=,=,<,>,≠}
<attribute name> is the name of the attribute of R
<constant value> is a constant value from the attribute domain
Clauses can be arbitrarily connected by the Boolean operators AND, OR and NOT to
form a general <select condition>

Example:Consider the relation PERSON. Display details of persons having age less than or equal to 30.

PERSON

PERSON_ID	NAME	AGE	ADDRESS
1	SANJAYA PRASAD	35	MODI NAGAR
2	SHARAD GUPTA	30	MAYUR VIHAR
3	VIBHU DATT	36	NEW DELHI

Answer:

□ (PERSON)
age<=30

PERSON_ID	NAME	AGE	ADDRESS
2	SHARAD GUPTA	30	MAYUR VIHAR

Example: Select the tuples for all employees who either work in department 4 and make over $25000 per year, or work in department 5 and make over$30000.

□(EMPLOYEE)
(dno=4 AND sal>25000) OR(dno=5 AND sal>30000)

❖ **Properties of Selection operation:**

- The number of tuples in selection relation is always less than or equal to the number
  i.e. | □ C (R) | ≤|R| where C is the condition.
- The number of attributes in selection relation □ C (R) is equal to the number of attributes in relation R.
  i.e. Degree(□ C (R))=Degree(R)

- Select operation is commutative .
  i.e. $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$ where C1 and C2 are any two conditions
- A cascade of select operations are combined into a single select operation with a conjunctive (AND) condition.
  i.e. $\sigma_{C1}(\sigma_{C2}(\ldots(\sigma_{Cn}(R))\ldots) = \sigma_{C1\ AND\ C2\ AND \ldots Cn}(R)$
  where C1, C2, …,Cn are n number of conditions

❖ **Projection(**

❖ The projection operation works on a single relation R and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates and it is denoted by the following symbol

$$\pi_{<attribute\ list>}(R)$$

❖ Example: Consider the "PERSON" relation and display only the names of persons.

π_{NAME}(PERSON)

NAME
SANJAYA PRASAD
SHARAD GUPTA
VIBHU DATT

❖ Display the name and age of "PERSON" relation.

$\pi_{NAME,\ AGE}$(PERSON)

NAME	AGE
SANJAYA PRASAD	35
SHARAD GUPTA	30
VIBHU DATT	36

❖ Example: Consider the " EMP" table as follows:

EMPNO	NAME	AGE	SALARY
123	SHAM	23	7500
124	KARAN	43	10000
125	RAHUL	23	10000

❖ Display age & salary from the above table.

π_{AGE}(PERSON)

AGE
23
43

π_{SALARY}(PERSON)

SALARY
7500
10000

- **Properties of Projection:**
  - The number of tuples in a relation resulting from a Project operation is always less than or equal to the number of tuples in R.

    $|\pi_{<A1,A2,An>}(R)| \leq |R|$
  - Degree of the project operation is equal to the number of attributes in <attribute list>

  - If the project list is a super key of R i.e. it includes some key of R, then the resulting relation has the same number of tuples as R.
  - If the project list includes only non key attributes of R, duplicate tuples are likely to occur, then the project operation removes any duplicate tuples and so the result of the project operation is a set of tuples, and hence a valid relation.
    This is known as **duplicate elimination**.

- **Rename( ) operation:**
  - Rename operation can rename either the relation names or the attribute names
  - or both.
  - The general rename operation when applied to a relation R of degree n is denoted by any of the following three forms:

  - $\rho_{S(A1,A2,\ldots,An)}(R)$ It renames both relation R and it's attributes by S and A1,A2,…,An OR

    $\rho_{S}(R)$ It renames the relation R by S only OR

    $\rho_{(A1,A2,\ldots,An)}(R)$ It renames the attributes of R by A1,A2,…,An only

- **Binary relational operations in relational algebra**
  - Cross join (Cartesian or Cross Product)
  - Join
  - Theta join
  - Equi join
  - Non equijoin
  - Self join
  - Natural join
  - Semi join

❖ **Join operations(X):(R1X$_{\text{<join condition>}}$ R2)**

- The join operation/operator allows us to combine any two relations R1 and R2 to form a single new relation.
- The join operation is used to combine related tuples from two relations into single tables.
- This operation is very important for any relational database with more than a single relation , because it allows us to process relationships among relations.
- The join operation requires that both the joined relations must have at least one domain compatible attributes.
- **Theta join**: It is a Cartesian product of two relations(tables) R1 and R2 followed by a restriction operation on the resultant relation(table).A join or theta join is denoted by the following symbol

  R1X $_{\text{<join condition>}}$R2=□□□(R1XR2)

  <join condition>

- The <join condition> or predicate P is of the form R1.Ai op R2.Bj, where op or operator is one of the comparison operators {=,<,≤,>,≥,≠}
- **Equijoin**: The join that uses only comparison operator equality (=) in <join condition> is called an equijoin , which contains two identical columns from the relations being joined. It is given by the following symbol: **R1X $_{\text{R1.Ai=R2.Aj}}$R2=□□□(R1XR2)**

  R1.Ai=R2.Aj

- **Non equijoin:** The join that uses any comparison operator other than equality operator in <join condition> is called as non equijoin
- **Self join:** When a relation (or table) is joined to itself using only equality operator in <join condition>, we call it as self join.

- **Natural join:** It is an equijoin with one of the two identical columns being eliminated.

- **Semi join:** It defines a relation that contains the tuples of R1 that participate in the natural join of R1 with R2.
  - Semi join decreases the number of tuples that need to be handled to form the join.
  - It is particularly useful for computing joins in distributed systems

- **Inner join**: An inner join is a type of match and merge operation defined formally as a combination of Cartesian product and Selection and it is used to combine data from multiple relations so that related information can be presented in a single table. Equijoin and Natural joins are known as inner joins.
- **Outer join**: An outer join is a type of equijoin that is used to show all data from one table, even if corresponding data is not found in the second table.
- Outer joins are commonly used with tables having 1:M relationships.
- There are three types of outer joins:
    - Left outer join
    - Right outer join
    - Full outer join
- **Left outer join:** The left outer join 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 from the right relation , and adds them to the result of the natural join.
- **Right outer join**: The right outer join takes all tuples from the right relation that did not match with any tuple in the left relation, pads the tuples with null values for all other attributes from the left relation, and adds them to the result of the natural join.
- **Full outer join**: It does both the operations of left outer join and right outer join , padding tuples from 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 additing them to the result of the join.
- **n-way join**: It is the natural join or equijoin operation among multiple tables.
- **Join selectivity:** It is the ratio of expected size of join to the maximum size |R1||R2|
- **Properties of join:**
    - Degree of R1 join R2 is less than or equal to sum of degrees of R1 and R2
      Degree(R1X $_{\text{<join condition>}}$R2) ≤ Degree(R1)+ Degree(R2)
    - Cardinality of R1 join R2 lies between 0 to |R1|.|R2|
      0≤ |R1X $_{\text{<join condition>}}$R2| ≤ |R1||R2|
- **Illustration of Joins through example :**Let us consider the following two tables named as 'employee' and 'ft_works' as follows:

Employee-name	street	city
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death valley
Williams	Seaview	Seattle

employee-name	branch-name	salary
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

- Find the equijoin, natural join, semi join, left outer join, right outer join and full outer join of the above two tables.

- Equijoin of the above two tables will be the following table :

Employee-name	street	city	Employee-name	branch-name	salary
Coyote	Toon	Hollywood	Coyote	Mesa	1500
Rabbit	Tunnel	Carrotville	Rabbit	Mesa	1300
Williams	Seaview	Seattle	Williams	Redmond	1500

- Natural join of above two tables will be the following table:

Employee-name	street	city	branch-name	salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500

- Left outer join of 'employee' and 'ft_works' will be the following table:

Employee-name	street	city	branch-name	salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death valley	null	null

- Right outer join of 'employee' and'ft_works' will be the following table:

Employee-name	Street	city	branch-name	salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Gates	Null	null	Redmond	5300

- Full outer join of 'employee' and 'ft_works' will be the following table:

Employee-name	street	city	branch-name	salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death valley	null	null
Gates	Null	null	Redmond	5300

- Semi join of the two tables:

Employee-name	street	city
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Williams	Seaview	Seattle

- **Division operation(/):**
  - The division operation defines a relation over the attributes C that consists of the set of tuples from R1 that match the combination of every tuple in R2
  - The division operator (/) divides a dividend relation R1 of degree m+n by a divisor relation R2 of degree n, and produces a resultant relation of degree m.
  - In the formulation of the division operation, the tuples in the denominator relation restrict the numerator relation by selecting those tuples in the result that match all values present in the denominator.

- **Properties of division operation:**
  - Degree(R1/R2)=Degree(R1)-Degree(R2)
  - The division operation can be expressed as a sequence of X, -} {□□□
- **A complete set of relational algebra operations:**
  - ❖ If any relational algebra operations can be expressed as a sequence of operations from the set {□□□□ U, -, X} then this set called a complete set.
  - ❖ Examples of complete set:
    - R1∩R2=(R1UR2)-((R1-R2)U(R2-R1))
    - R1X
    - A natural join can be specified as a cartesian product preceded by rename and followed by select and project operations
- **Illustration of division operation through example:**
- **Let R1=**

A	B
A1	B 1
A1	B2
A2	B1
A3	B1
A4	B2
A5	B1
A3	B2

- **And R2=**

B
B1
B2

- **R1/R2= AB/B=A=**

A
A1
A3

B1 in R2 relates to A in R1=(A1,A2,A3,A5)
B2 in R2 relates to A in R1=(A1,A4,A3)
(A1,A2,A3,A5) ∩(A1,A4,A3)=(A1,A3)=A

- **If R2=**

B
B1

- **Then R1/R2=AB/B=A =**

A
A1
A2
A3
A5

B1 in R2 has the values in A in R1 =(A1,A2,A3,A5)

- **If R2=**

A

- **Then R1/R2 =AB/A=B=**

B
B1
B2

- **If R2=**

B

- **Then R1/R2= AB/B=A=**

A
A1
A2
A3
A4
A5

The relational calculus:

- **Definition:** The relational calculus is a formal non-procedural language that uses predicates(or conditions) to write declarative expressions to specify a retrieval request specifying what is to be retrieved rather than how to retrieve it.
- **Importance of relational calculus**
- It has a firm basis in mathematical logic
- The SQL for RDBMS has some of its foundations in the tuple relational calculus
- Any retrieval that can be specified in the basic relational algebra can also be specified in relational calculus, and vice versa; in other words, the expressive power of two languages is identical
- **Relational complete language:**
- **Definition:** A relational query language L is considered relationally complete if we can express in L any query that can be expressed in relational calculus.
- **Classifications/Categories of relational calculus**
    - Tuple relational calculus(TRC) (Codd-1972)
    - Domain relational calculus(DRC) (Lacroix and Pirotte-1977)
- **TRC** : It finds tuples for which a predicate(or condition) is true. A tuple variable(t) is a variable that 'range over' a named relation i.e., a variable whose only permitted values are tuples of the relation
- A TRC query is of the form {t|P(t)} or {t|COND(t)} where P is the predicate and COND(t) is a conditional expression involving t and t is a tuple variable.
- **General expression in tuple relational calculus :**
- A general expression of the tuple relational calculus is of the form

  {t1.Aj, t2.Ak,…,tn.Am|COND(t1,t2,...,tn,...)}

  where Ai is an attribute of the relation on which tuple variables ti ranges and COND is a conditional formula (or WFF –well formed formula in mathematical logic).
- **Atom:** An atom is defined as of the following forms
    - R(ti)-range of the tuple variable ti as the relation whose name is R
    - ti.Aop tj.B where op={=,<,<=,>,>=,≠}, A and B are the attributes of the relation on which ti and tj ranges.
    - ti.A op c or optj.B where c is a constant value, A and B are the attributes of the relation on which ti and tj ranges and op={=,<,<=,>,>=,≠}, .
- **Formula(or Condition formula or Well formed formula-WFF)**

- A formula/WFF/conditional formula is made up of one or more atoms connected by the logical operators AND, OR , and NOT and is defined recursively as follows:
    - Every atom is a formula
    - If F1 and F2 are two formulae then F1ANDF2, F1ORF2, NOT F1, and NOT F2 are formulae
    - If F is a formula then (there exist t)(F)is a formula
    - If F is a formula then (for all t)(F) is a formula
- Truth value of an atom: Each of the atoms evaluates to either TRUE or FALSE for a specific combinations of tuples, this is called as the truth value of an atom
- The following are the truth tables of NOT, OR, AND , there exist and for all.

F	NOT F
T	F
F	T

F1	F2	F1OR F2
T	T	T
T	F	T
F	T	T
F	F	F

F1	F2	F1AND F2
T	T	T
T	F	F
F	T	F
F	F	F

F	(there exist t)(F)	(for all t)(F)
T	T	T
F	F	F

- **Quantifiers**: Quantifiers are used with formulae to tell how many instances the predicate
  (condition) applies to.
    - There are two quantifiers:
        - The existential quantifier(there exist): The existential quantifier is used in formulae that must be true for at least one instance.
        - The universal quantifier(for all): The universal quantifier is used in statements about every instance
- **Bound variables**: A tuple variable t is said to be bound if it is quantified by 'there exist'
  and 'for all'.

- **Free variables**: Tuple variables that are not bound are known as free variables. The only free variables in a relational calculus expression should be those in the left side of the bar(|).
- **Rules for free and bound variables**:
  - We define a tuple variable in a formula as free or bound according to the following rules:
  - An occurrence of a tuple variable in a formula F that is an atom is free in F.
  - An occurrence of a tuple variable t is free or bound in a formula made up of logical connectives(F1 AND F2), (F1 OR F2), NOT(F1) , NOT(F2) depending on whether it is free or bound in F1 or F2 if it occurs in either.
  - All free occurrences of a tuple variable t in F are bound in a formula F^1 of the form F^1=(there exist t)(F) or F^1=(for all)(F).
- **Closed WFF and Open WFF:** A WFF in which all variables references are bound is called closed WFF and A WFF that is not closed is called is called an open WFF.i.e.one that consists of at least one free variable reference.
- **Transforming the universal and existential quantifiers:**
  - It is possible to transform a universal quantifier into an existential quantifier , and vice versa, to get an equivalent expression. One general transformation can be described informally as follows:
    - Transform one type quantifier into the other with negation(preceded by not)
    - AND and OR replace one another
    - A negated formula becomes un negated
    - An un negated formula becomes negated
- **Safe expression**: A safe expression in relational calculus is one that is guaranteed to yield a finite number of tuples as its result; otherwise the expression is called unsafe. An expression is said to be safe if all values in its result are from the domain of the expression.
- Example : {t|NOT(EMPLOYEE(t)} is not safe i.e unsafe.
- **The domain relational calculus(DRC) (or Domain calculus):** In domain relational calculus , domain variables take their values from domain of attributes rather than tuple of relations.
- **An expression of the domain calculus is of the form**

  {x1,x2,...,xn| COND(x1,x2,...)}

  where x1,x2,x3,...,xn ,..... are domain variables that range over domains of attributes, and

  COND is a condition or formula or WFF of the domain relational calculus.
- **Atom:** An atom of a DRC is one of the following forms:
  - An atom of the form R(x1,x2,…,xj)
  - An atom of the form xi op xj
  - An atom of the form xi op c or c op xj
  - Where R is the name of a relation of degree j and each xi and xj are the domain

    variables and op={=,<,≤,>,≥,≠}= comparision operators and c is a constant
- **Formula (or conditional formula or Well formed formula-WFF):**
  - An atom is a formula
  - If F1 and F2 are formulae then F1 AND F2, F1 OR F2 , NOT F1 and NOT F2 are

also formulae

- If F is a formula with domain variable x, then (there exist x)(F) and (for all x)(F)

  are also formulae

- **Free and bound variables**: The rules concerning free and bound variables given for the tuple relational calculus is same as that of domain relational calculus.

- **Examples of tuple relational calculus:**
- Find all employees whose salary is above $50,000.
  Ans: {t|EMPLOYEE(t) AND t.salary>50000}
- Retrieve the first and last names of employees whose salary is above $50,000.
  Ans:{t| t.fname, t.lname| EMPLOYEE(t) AND t.salary>50000}
- Retrieve the birth date and address of the employee (or employees) whose name is 'John B. Smith'.
  Ans:{t.bdate,t.address| EMPLOYEE(t) AND t.fname='John' AND t.mname='B' AND t.lname='Smith'.
- Find all the books whose price is above 100.
  Ans:{t|BOOK(t) AND t.price>100}
- Retrieve the details of all books(TITLE, AUTHOR and NAME) which were published by 'Kalyani' and whose price is greater than 100.
  Ans:{t.TITLE,t.AUTHOR|BOOK(t) AND t.publisher='Kalyani' AND t.price>100}
- **Examples of domain relational calculus:**
- Find the branch name, loan number, and amount for loans of over $1200.
- Ans:{<b,l,a>|<b,l,a> belongs to loan AND a>1200}.
- Find all loan numbers for loans with an amount greater than $1200.
- Ans: {<l>| there exist b,a(<b,l,a> belongs to loan AND a>1200}.
- Find the names of all customers who have a loan from the Perryridge branch and find the loan amount.
- Ans:{<c,a>| there exist l (<c,l> belongs to borrower AND there exist b(<b,l,a> belongs to loan AND
  b=" Perryridge"}.

Relational database design by ER to Relational Mapping:

Step1: Mapping of regular entity types
Step2: Mapping of weak entity types
Step3: Mapping of binary 1:1 relation types
Step4: Mapping of binary 1:N relation types
Step5: Mapping of binary M:N relation types
Step6: Mapping of multi valued attributes
Step7: Mapping of N-ary relationship types

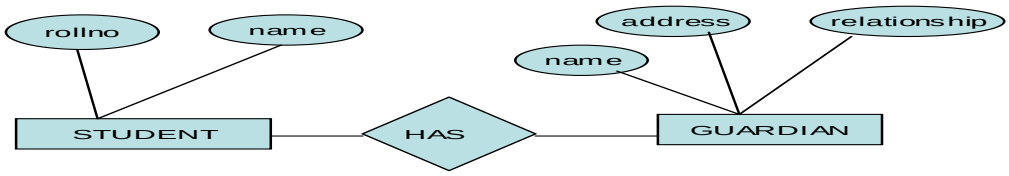
Step1: Mapping of regular entity types:

- For each regular(strong) entity type E in the ER schema, create a relation R that includes all
  the simple attributes of E.
- Choose one of the key attributes of E as the primary key for R. If the chosen key of E is
  composite , the set of simple attributes that form it will together form the primary key of R.
- Example: STUDENT(ROLLNO,NAME,ADDRESS), FACULTY(ID,NAME,ADDRES,BASIC-

SAL), COURSE(COURSE_ID, COURSE_NAME, DURATION) and DEPARTMENT(DNO,DNAME) are the tables for the corresponding strong entities 'STUDENT', 'FACULTY', 'COURSE', and 'DEPARTMENT'

Step2: Mapping of weak entity types:

- For each weak entity type W in the ER schema we create another relation R that contains all simple attributes of W as attributes of R.
- If E is an owner(strong) entity of W then key attribute of E is also included in R and this key attribute of R is set a a foreign key attribute of R
- Now the combination of the primary key attribute of owner entity type and partial key of weak entity type will form the key of the weak entity type.
- Example:



- GUARDIAN(name, address, relationship) is a weak entity whereas STUDENT(rollno, name, address) is a strong entity related by the relationship 'has'. Now according to rules of step2

we can have the mapped table of 'GUARDIAN' as GUARDIAN(rollno,name, address, relationship), where the primary key=(rollno, name) , foreign key = rollno which refers to the

primary key =rollno of strong entity 'STUDENT'.

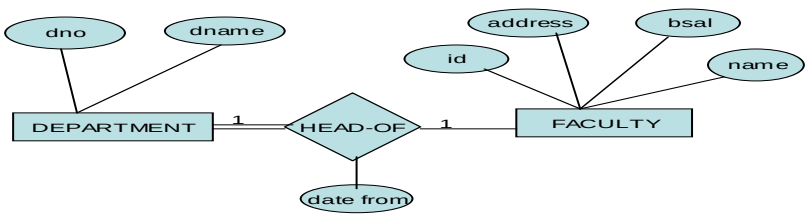
Step3: Mapping of binary 1:1 relation types: For each 1:1 relationship type R in the ER diagram

involving two entities E1 and E2 we choose one of entities(say E1) preferably with total participation and add primary key attribute of another entity E2 as a foreign key attribute

in entity E1. We will also include all the simple attributes of relationship type R in E1 if any.

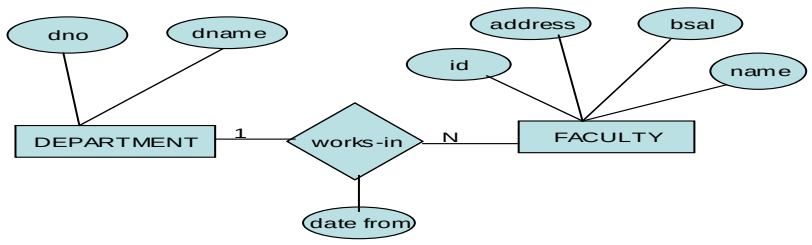
Example: In the following ER diagram (fig. below)' DEPARTMNT' table is converted to

DEPARTMENT(dno,dname, head_id(=id=fk), date from).

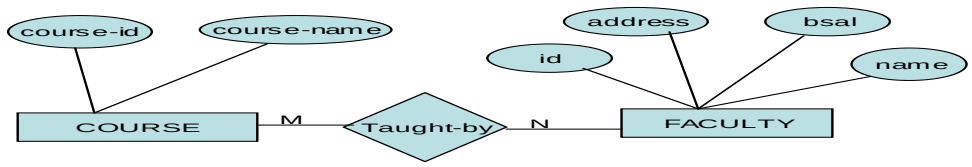


Step4: Mapping of binary 1:N relation types: For each 1:N relationship type R involving two entities E1 and E2 , we identify the entity type (say E1) at the N-side of the relationship type R and include the primary key of the entity on the other side of the relation(say E2) as a

foreign key attribute in the entity E1. We include all simple attributes(or simple components of a composite attributes of R if any in the table of E1).
Example: From the ER diagram (given below) it is clear that 'FACULTY' can be converted to the following table according to step4: FACULTY(id(pk), name, address, bsal, dno(fk)).



Step5: Mapping of binary M:N relation types: 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 attributes in S. Any simple attributes of the M:N relationship type(or simple components of composite attribute) are also included as attributes of S.
Example: A new table TAUGHT_BY will be created as follows
TAUGHT_BY(id,course-id)



Step6: Mapping of multi valued attributes: For each multi valued attribute 'A', we create a new relation R that includes an attribute corresponding to plus the primary key attribute K of the relation that represents the entity type or relationship type that has an attribute. The primary key of R is then the combination of A and K.
R(K(PK),A(MVA),A1) converted to R1(K,A) and R2(K,A1)
Example: If a STUDENT(rollno,name,phoneno) is an entity having 'phoneno' as the multivalued attribute then we will create a table PHONE(rollno,phoneno) where the combination (rollno, phoneno)
is the primary key and we will also create another table STUDENT(rollno,name) where rollno is the primary key.
Step7: Mapping of N-ary relationship types:

- For each n-ary relationship type R, create a new relationship S to represent R .
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
- Also include any simple attributes of the n-ary relationship type(or simple components of composite attributes) as attributes of S.

Relational database design by EER diagram to relational database:
(Conversion of Generalization/Specialization hierarchy to tables)

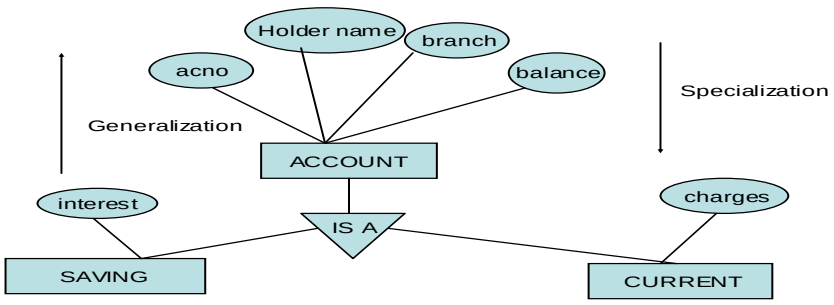
Example: A bank has an account entity sets. The account of the bank can be of two types:

- SAVING ACCOUNT and
- CURRRENT ACCOUNT

- The statement as above represents a specialization/generalization hierarchy, which can be shown by EER diagram as shown in figure below:
- A simple way for conversion may be to decompose all the specialized entities into tables in case they are:

  - **Disjoint:** When A/Cs are disjoint, then the tables are :
    - **SAVING(acno,holder name, branch, balance, interest)**
    - **CURRENT(acno, holder name, branch, balance, charges)**

  - **Overlapping:(Joint A/C):**When A/Cs are joint then the tables can be
    - **ACCOUNT(acno, holder name, branch, balance)**
    - **SAVING(acno, interest)**
    - **CURRENT(acno,charges)**



Module 2 (16 hours)

FUNCTIONAL DEPENDENCY(FD) (Maier,1983)

Definition: If X and Y are attributes of a relation R, then Y is said to be functionally dependent on X (or Y is functionally determined by X or X functionally determines Y), denoted by X->Y, if each value of X is associated with exactly one value of Y so that for any two tuples t1 and t2 in relation state r of R, we have

- X->Y => (t1[X]=t2[X] => t1[Y]=t2[Y])
- L.H.S.(=X) is known as determinant
- R.H.S.(=Y) is known as determined

Examples-FUNCTIONAL DEPENDENCY(FD)

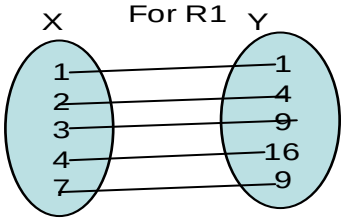
R1

X	Y
1	1
2	4
3	9
4	16
7	9

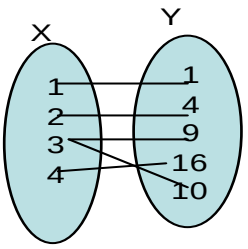
R2

X	Y
1	1
2	4
3	9
4	16
3	10

- In R1, X->Y, because it is one-one relation
- In R2, X->Y is not satisfied because it is one-many relation



R1:One-to-One relationship between data for X and Y

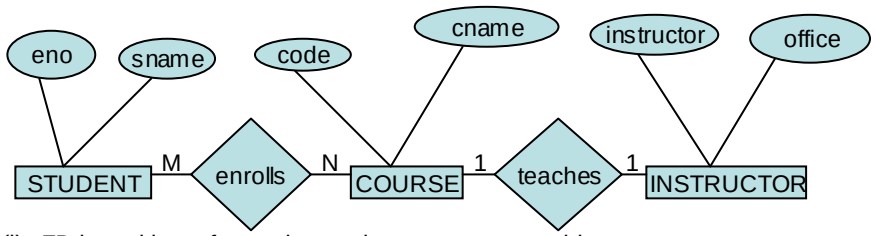


R2:one-to-many relationship between data for X and Y

- X->Y (i.e. Y is functionly dependent on X) <=>There exists one-to-one and many-to-one relationships (associations) between data values of attributes X and Y
- Yis notfunctionally dependent on X if there exists one-to-many and many-to-many relationships (associations) between data values of attributes X and Y
- A functional dependency is a constraint between two sets of attributes from the database(i.e. Data inter-relationship in a relation or relationship between attributes)
- FD is a property of the semantics or meaning of the attributes.
- FD is a property of a relational schema(intension) and not a property of a particular instance of the schema(extension).
- If X->Y in R, this does not say whether or not Y->X in R
- **FD in relationships**: If X and Y be the primary keys of two different entities or tables and the two entities have relationships(1:1, M:1, 1:N, M:N), then we must have the following conclusions:
  - For one-to-one (1:1)relationships: X->Y and Y->X
  - For many-to-one (M:1)relationship(with X on many side): X->Y but not Y->X
  - For one-to- many(1:M)and many-to-many(M:N) relationship: No FD exists.

FD in ER diagrams

- ER diagram for student course teacher



(i) FD in entities -: for student entity, eno->sname,address
for course entity, code->cname
for instructor entity, instructor-> office

(ii)FD in relationships-:for enrolls relationships, No FD exists as it is many-many
for teaches relationships , code->instructor and instructor-> office

- **FD in ER diagrams**:

- **Exercise:** Below is an instance of R(A1,A2,A3,A4). Choose the FD which may hold on R. mm1.A4->A1 , 2.A2A3->A4, 3.A2A3->A1 .

A1	A2	A3	A4
1	2	3	4
1	2	3	5
6	7	8	2
2	1	3	4

Solution: 1.A4->A1 is incorrect: The 1st and 4th tuple violates it.
2. A2A3->A4 is incorrect: The1st and 2nd tuple violates it.
3. A2A3 -> A1 is correct.

REDUNDANCY AND ASSOCIATED PROBLEMS

- **Anomalies(or database anomalies):** Relations(or tables) that that have redundant data may have problems of wastage of memory space called as anomalies.
- **Classification (or types) of anomalies:**
  - Update(or modification) anomaly
  - Insertion anomaly
  - Deletion anomaly
    - **Update anomaly**: This anomaly is caused due to the data redundancy. Redundant information makes updates more difficult. An update anomaly results in data inconsistency.
    - **Insertion anomaly**: This anomaly is caused due to the inability to represent certain information.
    - **Deletion anomaly**: This anomaly is caused due to the loss of information.
- Illustration of anomalies through example: Consider the relation "STUDENT" as shown in figure below:

Eno	sname	address	code	cname	instructor	Offic

						e
0511234 5	RAHUL	RANCHI	MCS-011	PROBLEM SOLUTION	NAYAN KUMAR	102
0511234 5	RAHUL	RANCHI	MCS-012	COMPUTER ORGANIZATI ON	ANURA G SHARM A	105
0511234 5	RAHUL	RANCHI	MCS-014	SAD	PREETI ANAND	103
0511341	APARAN A	GURGAO N	MCS-014	SAD	PREETI ANAND	103

- **Update anomaly:** Changing the name of the instructor of MCS-014 would require that all tuples containing MCS-014 would require that all tuples containing MCS-014 enrolment information be updated. If for some reason , all tuples are not updated , we might have a database that gives two names of instructor for the subject MCS-014 which is inconsistent information. This problem is called update anomaly.
- Insertion anomaly: If one wanted to insert the code and name of a new course in the database 'STUDENT', it would not be possible until a student enrolls in that course. Information about a new student can't be inserted in the database until the student enrolls in a course. These problems are called insertion anomalies.
- Deletion anomaly: In 'STUDENT' relation, if we delete the tuple corresponding to student'05011341' enrolled for MCS-014, we will lose relevant information about the student like eno, snme and address of this student. Deletion of tuple having sname "RAHUL" and cno MCS-012" will result in loss of information that MCS-012 is named computer organization having an instructor "Anurag Sharama" , whose office number is 105. These are called deletion anomalies.

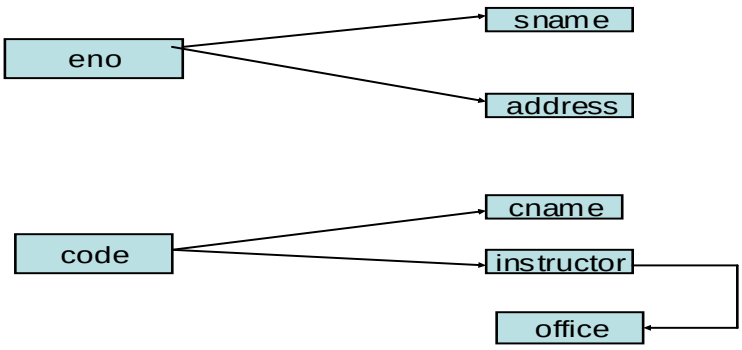
- The anomalies arise primarily because the relation "STUDENT" has information about students as well as subjects.
- Solution to the database anomalies: One solution to the anomaly problems is to decompose the relation into two or more small relations. The basis of this decomposition is determined by data inter-relationship. Data inter-relationship is determined by the concept of functional dependency. The root cause of the presence of anomalies in a relation is determined by the components of the key and non-key attributes.
- Hence {eno}->{sname,address}, {code}->{cname,instructor},{instructor}->{office}
- Now the smaller tables are:

Eno	sname	address
05112345	RAHUL	RANCHI
0511341	APARANA	GURGAON

Code	cname	instructor
MCS-011	PROBLEM SOLUTION	NAYAN KUMAR
MCS-012	COMPUTER ORGANIZATION	ANURAG SHARMA
MCS-014	SAD	PREETI ANAND

insructor	office
NAYAN KUMAR	102
ANURAG SHARMA	105
PREETI ANAND	103

Normalization involves decomposition of a relation into smaller relations based on the concept of functional dependencies to overcome undesirable anomalies



Pictorial representation of FDs

- **Trivial and Non-trivial FDs**
  - X->Y is trivial iff Y $\subseteq$X otherwise X->Y is non-trivial
  - Non trivial FDs represent integrity constraints for the relation
- Main characteristics of F.D.'s that we use in normalization: Functional dependencies
  - have one-to-one relationship between attribute(s) on L.H.S. and R.H.S of the FD
  - holds for all time
  - are non-trivial
- **Closure of functional dependencies(F^+)**:

Definition: Closure of a set F of FDs is defined as the set F^+ of all FDs that include F as well as

all dependencies that can be inferred from F .If the functional dependency X->Y is inferred

from the set of functional dependencies F then we say that F logically implies(|=) X->Y, denoted by F|=X->Y. i.e.**F^+={X->Y| F|=X->Y}**

- **Armstrong's axioms(or Armstrong's inference rules) (Armstrong-1974)**
  - If X, Y and Z be the subsets of the attributes of a relation R, then Armstrong's axioms are as follows**:**
    - IR1. (**Reflexive or self determination rule**): If Y $\subseteq$X, then X -> Y or X->X
    - IR2. (**Augmentation rule**): If X -> Y, then XZ -> YZ or XZ->Y
      (Notation: XZ stands for X U Z)
    - IR3. (**Transitive**): If X -> Y and Y -> Z, then X -> Z
- **Inference rules using Armstrong's axioms are as follows:**
  - IR4**.** (**Decomposition or Projective rule**) :If X -> YZ, then X -> Y and X -> Z
  - IR5. (**Union or additive rule**): If X -> Y and X -> Z, then X -> YZ

- IR6. (**Psuedotransitivity rule**): If X -> Y and WY -> Z, then WX -> Z
- IR7.(**Composition rule**): If X->Y, Z->W, then XZ->YW
- IR8.(**Self accumulation rule**): If X->YZ, Z->W, then X->YZW

- **Proof of Armstrong's Axiom and their additional inference rules** :
  - **Prof of IR1**.(i) If $Y \subseteq X$ and for any two tuples t1 and t2 that exist in some relation instance r of R such that t1[X]=t2[X] => t1[Y]=t2[Y] , then X->Y must hold in r of R. (Proved)

    (ii) **Y is subset of X => X->Y**. If Y=X then **$X \subseteq$X=> X->X**. (Proved).
  - **Proof of IR2**.(By contradiction method):

    (i)Let us assume that X->Y holds in a relation instance r of R but XZ->YZ doesn't hold.

    then there must exist two tuples t1 and t2 in r such that

    (1)t1[X]=t2[X]

    (2)t1[Y]=t2[Y]

    (3)t1[XZ]=t2[X

    (4)t1[YZ] ≠ [YZ]

    X->Y => t1[X]=t2[X] =>t1[Y]=t2[Y]
    XZ->YZ does not hold good=> t1[XZ]=t2[XZ]=> t1[YZ] ≠t2 [YZ]

    From (1) and (3) we deduce

    (5)t1[Z]=t2[Z]

    From (2) and (5) we deduce

    (6)t1[YZ]=t2[YZ], contradicting (4)

    So **X->Y=>XZ->YZ** (Proved)
  - (ii) Let us assume that X->Y holds but XZ->Y doesn't hold . Then we must have:

    X->Y => t1[X]=t2[X] =>t1[Y]=t2[Y]

    XZ->Y does not hold good=> t1[XZ]=t2[XZ]=> t1[Y] ≠ t2[Y]

    (1)t1[X]=t2[X]

    (2)t1[Y]=t2[Y]

    (3)t1[XZ]=t2[XZ]

    (4)t1[Y] ≠t2[Y]

    (2) and (4) gives the contradiction.

    So **X->Y=>XZ->Y** (Proved)
  - **Proof of IR3**: Let us assume that X->Y and Y->Z both holds in relation instance r of R.

    Then for any two tuples t1 and t2 in r of R we must have t1[X]=t2[X] =>t1[Y]=t2[Y] and t1[Y]=t2[Y]=>t1[Z]=t2[Z]. Now we can write:

    t1[X]=t2[X]=>t1[Z]=t2[Z] => X->Z.

    Hence **X->Y, Y->Z=>X->Z** (Proved)
  - **Proof of IR4**: (1) X->YZ (Given)

    (2)YZ->Y (because: Y subset YZ =>YZ->Y , by reflexivity rule/IR1)

    (3)X->Y (because: X->YZ , YZ->Y=> X->Y, by transitivity rule/IR3)

    (4) X->YZ (Given)

    (5)YZ->Z (because: Z subset of YZ=> YZ->Z, by reflexivity rule/IR1)

    (6)X->Z (because: X->YZ, YZ->Z=>X->Z, by transitivity rule/IR3)

    From (3) and (6) it is clear that X->YZ=> X->Y, X->Z.(Proved).
  - **Proof of IR5**: (1)X->Y (Given)

    (2)X->Z (Given)

    (3) X->XY (because: X->Y =>XX->XY=>X->XY, by augmentation rule applied to (1) by X/IR2 and XX=X)

    (4)XY->YZ (because: X->Z =>XY->YZ, by augmentation rule applied

to (2) by Y/ IR2)

(5)X->YZ (because: X->XY, XY->YZ => X->YZ, by transitivity rule applied to (3) and (4)/IR3)

So **X->Y, X->Z=> X->YZ**. (Proved)

- **Proof of IR6**: (1) X->Y (Given)
  (2) WY->Z(Given)
  (3)WX->WY (because: X->Y=> WX->WY, by augmentation rule/IR2)
  (4)WX->Z(because: WX->WY, WY->Z=>WX->Z, by transitivity/IR3)
  So **X-> Y, WY->Z => WX->Z** (Proved)

- **Proof of IR7**: (1) X->Y (Given)
  (2) XZ->YZ (By augmentation rule/IR2 applied to (1) by Z)
  (3)Z->W (Given)
  (4)YZ->YW (By augmentation rule/IR2 applied to (3) by Y)
  (5)XZ->YW (By transitivity rule/IR2 applied to (2) and (4))
  So **X->Y, Z->W => XZ->YW** (Proved)

- **Proof of IR8**: Given X-> YZ => X->Y, X->Z (By decomposing rule/IR4)
  X->Z, Z->W=> X->W (By transitivity rule/IR3)
  X->YZ, X->W => X-> YZW (By union rule/IR5)
  X->YZ, Z->W => X->YZW (Proved)

- **Armstrong's Axioms(IR1,IR2, and IR3) are sound and complete**
- **Armstrong's Axioms are sound**

Given a set of functional dependencies F specified on a relation schema R, any dependency that we can infer from F by using Armstrong's axiom(IR1 through IR3) holds in every relation r of R that satisfies the dependencies in F.

- **Armstrong's Axioms are complete**

Using inference rules IR1 through IR3 (Armstrong's Axiom) repeatedly to infer dependencies until no more dependencies can be inferred results in the complete set of all possible dependencies that can be inferred from F.

Or

The set of dependencies can be determined from F by using only inference rules (IR1,IR2 and IR3)of Armstrong.

- **Solved Problems on Closure of functional dependencies(F^+):**

Q.1. If R=(A,B,C,D) and F={A->B, A->C,BC->D} then prove that F|=A->D.

Solution: {A->B,A->C}|=A->BC (By union rule-IR5)
{A->BC, BC->D}|=A->D (By transitivity rule-IR3)
=> F|=A->D =>A->D is in F^+. (Proved).

Q.2.If F={W->X, X->Y, W->XY} then find the closure of functional dependencies(F^+).

Solution: F^+ = { W->X, X->Y, W->XY, W->W, X->X, Y->Y, W->Y}
because W->W, X->X, Y->Y are obtained by reflexivity rule(self determination rule-IR1) and W->X, X->Y=> W->Y by transitivity rule(IR3). Then
F^+ = F U { Inference rules obtained from F}

={W->X, X->Y, W->XY} U { X->X, Y->Y,W->W, W->Y}
={W->W, X->X, Y->Y, W->Y, W->X, X->Y, W-> XY}.(Answer).

- **Closure of attribute X under set of FD F:**
  Definition: Closure of a set of attribute(s) X with respect to the set of functional dependencies F is the set X^+ of all attributes{ A1,A2,…,Am} that are functionally determined by X based on F such that X->Ai (Ai belongs to X^+) can be calculated by repeatedly applying Armstrong's axiom(IR1,IR2,and IR3).
- Lemma: F|=X->Y <=> $Y \subseteq X$
- ***Attribute Closure Algorithm(T*o compute the closure of attribute X (X^+) under fd F)**

```
X+:= X ;// according to reflexivity
repeat
{
        oldX+ = X+;
          for (each fd Y →Z ∈F )
        {
                if (Y ⊆ X+)
                    { X+ = X+ ∪ Z ;//according to   augmentation & transitivity
                     }
          }
  } until ( X+= oldX+)
```

- **Time complexity of *Attribute Closure Algorithm***
  - If a and f are the number of attributes and functional dependencies present in the set F and each FD in F involves only one attribute on R.H.S., then the inner for loop will be executed at most 'f' times, one for each FD in F, and each each such execution can take the time proportional to 'a' to check if one set is contained in another set.
  - The order of execution of the for loop is O(af)
  - The while loop can be repeated at most 'f' times and so the time complexity of the attribute closure algorithm is given by $O(af^2)$
- **Membership algorithm(Testing if an FD is in a closure)**
  Input: A set of functional dependencies F and the functional dependency X->Y
  Output: Is X->Y $\in F^+$ or not
  Compute X^+ using closure algorithm
  If $Y \subseteq X^+$ then X->Y $\in F^+$:=true else X->Y $\in F^+$:=false
- **Tricks for finding the keys from a set of functional dependencies:**
  - If an attribute never appears on the R.H.S. of any FD, it must be part of the key.
  - If an attribute never appears on the L.H.S. of any FD, but appears on the R.H.S. of any FD, it must not be the part of any key.
  - If the closure of any attribute(s) X in a relation R is equal to the set of all the attributes in the relation (X->R i.e $X^+ = R$), then X is super key of the relation.
  - The super key X is said to be the candidate key of the relation R if the closures of all the proper subsets of X is not equal to the set of all the attributes in the relation . (i.e. Candidate key is a super key whose all proper subsets are not unique)
    ($A \subseteq X$ => $A^+ \neq R$ but $A^+ \subseteq R$, where A is the proper subset of attribute X).
  - ***To find candidate keys using Closure of attributes***
    Question1: We are given a relation R(A,B,C,D,E) and the functional dependencies (FDs) amongst its attributes F = {A -> B, AC -> D, B -> E}

and asked to find the candidate key(s) of R.

Solution1: Find out the closure of each determinant (L.H.S.)of F . The determinants are {A}, {AC}, and{B}

To see if determinate A is a super key using attribute closure algorithm

X+ := {A}	X+ is {A}
oldX+:= {A}	
for A -> B, A $\subseteq$ X+	
so B is added to X+	X+ is now {A,B}
for AC -> D,	
AC is not a subset of X+	X+ is still {A,B}
for B -> E, B $\subseteq$ X+	
so E is added to X+	
oldX+ != X+ **repeat**	X+ is now {A,B,E}
oldX+:= {A,B,E}	
for A -> B,	
A $\subseteq$ X+, B is already in X+	
for AC -> D,	X+ is still {A,B,E}
AC is not a subset of X+	
for B -> E,	X+ is still {A,B,E}
B $\subseteq$ X+, E is already in X+	
X+ = oldX+ **stop**	X+ is still {A,B,E}

The closure of {A} over F is {A,B,E} which is a proper subset of R. Therefore, {A} is *not* a super key of R and *cannot* be a candidate key for R.

To see if determinate AC is a superkey using attribute closure algorithm

X+ := AC	X+ is {A,C}
oldX+:= {A,C}	
for A -> B,	
A is a subset of X+	
so B is added to X+	X+ is now {A,C,B}
for AC -> D,	
AC is a subset of X+	
so D is added to X+	X+ is now {A,C,B,D}
for B -> E,	
B is a subset of X+	
so E is added to X+	X+ is now {A,C,B,D,E}
oldX+ != X+ **repeat**	
oldX+:= {A,C,B,D,E}	
for A -> B,	
A is a subset of X+, B is already in X+	X+ is still {A,C,B,D,E}
for AC -> D,	

AC is not a subset of X+ for B -> E, B is a subset of X+, E is already in X+ X+ = oldX+ **stop**	X+ is still {A,C,B,D,E} X+ is still {A,C,B,D,E}

The closure of {A,C} over F is {A,B,C,D,E}=R which is not a proper subset of R. {A,C} is a super key for R and may be a candidate key for R.

To see if determinate B is a superkey using attribute closure algorithm

X+ := B	X+ is {B}
oldX+:= {B} for A -> B, A is a not a subset of X+ for AC -> D, AC is not a subset of X+ for B -> E, B is a subset of X+ so E is added to X+ oldX+ != X+ **repeat**	 X+ is still {B} X+ is still {B} X+ is now {B,E}
oldX+:= {B,E} for A -> B, A is a subset of X+ for AC -> D, AC is not a subset of X+ for B -> E, B is a subset of X+, E is already in X+ X+ = oldX+ **stop**	 X+ is still {B,E} X+ is still {B,E} X+ is still {B,E}

The closure of {B} over F is {B,E} which is a proper subset of R. Therefore, {B} is *not* a superkey of R and *cannot* be a candidate key for R.

Using attribute closure algorithm , I have found one superkey {A,C} for R.

A candidate key is a superkey such that no proper subset is a superkey within the relation. So, I need to check the proper subsets of the superkey. The proper subsets of {A,C} are

- the null set
- {A}
- {C}

Examining this possible superkeys, I note that:

- A candidate key, which may be chosen to be a primary key, cannot be null.
- I have already discovered above that {A} is not a superkey for R.

So I need to consider only {C} as a possible superkey.

Using attribute closure algorithm to see if C is a superkey

X+ := C	X+ is {C}
oldX+:= {C} for A -> B, A is a not a subset of X+	X+ is still {C}
for AC -> D, AC is not a subset of X+	X+ is still {C}
for B -> E, B is not a subset of X+	X+ is still {C}
oldX+ = X+ **stop**	

The closure of {C} over F is {C} which is a proper subset of R. Therefore, {C} is *not* a superkey of R and *cannot* be a candidate key for R.

I have discovered that no proper subsets of {A,C} are superkeys of R, so {A,C} is a candidate key for R.

I know that any attribute which is in **none** of the determinants in F *cannot* be a candidate key.

Are there any other candidate keys for R?

I know that any attribute which is in **none** of the determinants in F *cannot* be a candidate key.

So {D}, {E} and {DE} can't be the candidate keys. If we would like to see an exhaustive application of attribute closure algorithm ,we can find all the subsets of R and apply the algorithm to each subset. I have found only one candidate key {A,C} which I must select as the primary key for R. Therefore, the relational schema for R is R (A, C, [pk] B, D, E) .

Alternative method: We have : F = {A -> B, AC -> D, B -> E}. Since the attributes A and C never appears on the R.H.S of any FD in F, both A and C must be part of the key. Both attributes D and E never appears on the L.H.S. of any FD in F, but appears on the RHS of any FD, both D and E must not be the part of the key. As A , C are the part of the keys, both A and C can't be the keys but their combination {AC} may be the key. So we find the closure of the attribute {AC}as follows:

AC^+ = AC (as AC->AC, by reflexivity rule)

=ABC (as A->B, by augmentation rule)

=ABCD (as AC->D, by augmentation rule)

=ABCDE (as B->E, by augmentation rule)

=R

AC^+ =ABCDE=R => {AC} is a super key and can be a candidate key if all the proper subsets are not super keys (i.e. closure of all proper subsets of {AC} are not equal to R=ABCDE. All the proper subsets of {AC} are null set, {A}, {C}. Null set can't be a key because super key/candidate key can't be null. Now the closure of {A} is given below:

A^+ = A (as A->A)

=AB (as A->B)

=ABE (as B->E) ≠ R => {A} is not unique => {A} is not a super key.

C^+ = C (as C-> C) ≠ R => {C} is not unique => {C} is not a super key.

So $\{AC\}^+ = R$, $\{A\}^+ \neq R$, $\{C\}^+ \neq R$ i.e.{AC} is a super key such that all of its proper subsets are not super keys => {AC} is a candidate key.(Answer).

Question2: Let R(ABCDEFGH) satisfy the following functional dependencies: {A->B, CH->A, B->E, BD->C, EG->H, DE->F}.Which of the following FD is also guaranteed to be satisfied by R?

1. BFG --> AE
2. ACG --> DH
3. CEG --> AB

Hint: Compute the closure of the LHS of each FD that you get as a choice. If the RHS of the candidate FD is contained in the closure, then the candidate follows from the given FDs, otherwise not.

Solution2:FDs: {A->B, CH->A, B->E, BD->C, EG->H, DE->F}

1. BFG --> AE ???.**Incorrect:** BFG+ = BFGEH, which includes E, but not A.

2. ACG --> DH ???.**Incorrect:** ACG+ = ACGBE, which includes neither D nor H.

3. CEG --> AB ???.**Correct:** CEG+ = CEGHAB, which contains AB.

Question 3: Which of the following could be a key for R(A,B,C,D,E,F,G) with functional dependencies {AB->C, CD->E, EF->G, FG->E, DE->C, and BC->A}.1. BDF,2. ACDF, 3. ABDFG,4. BDFG

Solution 3:F={AB->C, CD->E, EF->G, FG->E, DE->C, and BC->A}

1. BDF ???.No. BDF+ = BDF

2. ACDF ???.No. ACDF+ = ACDFEG (The closure does not include B)

3. ABDFG ???.No. This choice is a super key, but it has proper subsets that are also keys. (e.g. BDFG+ = BDFGECA)

4. BDFG ???.BDFG+ = ABCDEFG

- Check if any subset of BDFG is a key:

- Since B, D, F never appear on the RHS of the FDs, they must form part of the key.
- BDF+ = BDF ← Not key.
- So, BDFG is the minimal key, hence the candidate key.

Question4: Consider R = {A, B, C, D, E, F, G, H} with a set of FDs F = {CD→A, EC→H, GHB→AB, C→D, EG→A, H→B, BE→CD, EC→B}. Find all the candidate keys of R **.**

Solution 4: F = {CD→A, EC→H, GHB→AB, C→D, EG→A, H→B, BE→CD, EC→B}

- First, we notice that:
  - EFG never appear on RHS of any FD. So, EFG must be part of ANY key of R
  - A never appears on LHS of any FD, but appears on RHS of some FD. So, A is not part of ANY key of R
  - We now see if EFG is itself a key…
  - EFG+ = EFGA ≠ R; So, EFG alone is not key
  - Checking by adding single attribute with **EFG** (except **A**):
  - BEFG+ = ABCDEFGH = R; it's a key [BE→CD, EG→A, EC→H]
  - CEFG+ = ABCDEFGH = R; it's a key [EG→A, EC→H, H→B, BE→CD]
  - DEFG+ = ADEFG ≠ R; it's not a key [EG→A]
  - EFGH+ = ABCDEFGH = R; it's a key [EG→A, H→B, BE→CD]
  - If we add any further attribute(s), they will form the superkey. Therefore, we can stop here searching for candidate key(s).
  - Therefore, candidate keys are: {BEFG, CEFG, EFGH}

Exercise 5: Consider R = {A, B, C, D, E, F, G} with a set of FDs F = {ABC→DE, AB→D, DE→ABCF, E→C}. Find all the candidate keys of R .

Solution 5: F = {ABC→DE, AB→D, DE→ABCF, E→C}

- First, we notice that:
  - G never appears on RHS of any FD. So, G must be part of ANY key of R.
  - F never appears on LHS of any FD, but appears on RHS of some FD. So, F is not part of ANY key of R
  - G+ = G ≠ R So, G alone is not a key!
  - Now we try to find keys by adding more attributes (except F) to G
  - Add LHS of FDs that have only one attribute (E in E→C):
  - GE+ = GEC ≠ R
  - Add LHS of FDs that have two attributes (AB in AB→D and DE in DE→ABCF):
  - GAB+ = GABD
  - GDE+ = ABCDEFG = R; [DE→ABCF] It's a key!
  - Add LHS of FDs that have three attributes (ABC in ABC→DE), but not taking super set of GDE:
  - GABC+ = ABCDEFG = R; [ABC→DE, DE→ABCF] It's a key!
  - GABE+ = ABCDEFG = R; [AB→D, DE→ABCF] It's a key!
  - If we add any further attribute(s), they will form the superkey. Therefore, we can stop here.
  - The candidate key(s) are {GDE, GABC, GABE}

Exercise 6: Consider R = {A, B, C, D, E} with a set of FDs F = {AB→DE, C→E, D→C, E→A}

And we wish to project those FDs onto relation S={A, B, C}.Give the FDs that hold in S

- **Hint: We need to compute the closure of all the subsets of {A, B, C}, except the empty set and ABC. Then, we ignore the FDs that are trivial and those that have D or E on the RHS**

Solution6: R = {A, B, C, D, E}F = {AB→DE, C→E, D→C, E→A}S={A, B, C}

- A+ = A
- B+ = B
- C+ = CEA [C→E, E→A]
- AB+ = ABDEC [AB→DE, D→C]
- AC+ = ACE [C→E]
- BC+ = BCEAD [C→E, E→A, AB→DE]
- We ignore D and E.
- So, the FDs that hold in S are:
- {C→A, AB→C, BC→A}
- (Note: BC→A can be ignored because it follows logically from C→A)

Exercise7: If R=(A, B, C,D,E), F={B->CD, D->E, B->A, E->C} then check for the followings?

1. Is B->E in F^+?, 2. Is D a key for R?, 3. Is AD a key for R?, 4. Is AD a candidate key for R?

5.Is ADE a candidate key for R?

Solution 7.1. B^+=ABCDE=R (B->CD, D->E, B->A) =>B->ABCDE=> B->E $\in F^+$

2.D^+=DEC(D->E, E->C) $\neq$ R=> D is not a key for R.

3.AD^+ =ABCDE=R (AD->B, B->CD, D->E) => AD is a key for R.

4. AD^+ =ABCDE=R, A^+= A$\neq$ R and D^+= DEC(D->E, E->C) $\neq$ R
=>AD is a candidate key for R.

5.{AD} is a key => {ADE} is a super key but not a candidate key.

Exercise8: If R(ABCDE) be a relation with a set of functional dependencies F(A->BC, CD->E,
B->D, E->A) then find all candidate keys of R.

Answer8: Here all the attributes of R are present in both L.H.S. and R.H.S of F. Hence we can't compute all the candidate keys easily using attribute closure algorithm.

A^+ = A(as A->A)=ABC(as A->BC)=ABCD(as B->D)=ABCDE(as CD->E)=R
⇨ {A} is the candidate key.

CD^+ = CD(as CD->CD) =CDE(as CD->E) = CDEA (as E->A)=CDEABC(as A->BC)=R ,

C^+=C $\neq$ R and D^+ = D$\neq$ R => {CD} is a candidate key.

B^+ = B(as B->B) = BD(as B->D) $\neq$ R => {B} is not a candidate key.

E^+ = E(as E->E) = EA(as E->A) = EABC (as A->BC) = EABCD(as B->D)=R
⇨ {E} is a candidate key.
⇨ E->ABCDE

B->D => BC-> CD(by augmentation rule) .

CD-> E , E-> ABCDE => CD-> ABCDE (by transitivity rule)

Again BC->CD, CD->ABCDE => BC->ABCDE (by transitivity rule)=> BC^+=ABCDE=R

⇨ {BC} is a candidate key as B^+ = B(as B->B) = BD(as B->D) ≠ R and C^+=C ≠ R

.

So {A}, {BC}, {CD} and {E} are the candidate keys of R.

Exercise9: F={ A->BC, CD->E, E->C, D->AEH, ABH->BD, DH->BC}. Test whether F|=BCD->H or not?

Answer9: BCD^+ = BCDE (as CD->E)

=BCDEAH (as D->AEH)

=ABCDEH =R =>H $\subseteq$ BCD^+ (=ABCDEH) => F|=BCD->H=> BCD->H$\in$ F^+

Exercise 10:If R(ABCDEH) and F={A->BC, CD->E, E->C, D->AEH, ABH->BD, DH->BC}, then find the candidate keys of R.

Answer10: A^+ = ABC (as A->BC)≠R

CD^+ = CDE (as CD->E)

=CDAEH (as D->AEH)

=ABCDEH (as DH->BC)=R and C^+≠R, D^+=DAEH=DABCEH=ABCDEH=R

{CD} is the super key and {D} is the candidate key

Again A->BC and D->AEH => AD->ABCEH => AD^+ = ABCDEH=R

Since A^+ = ABC (as A->BC)≠R and D^+=DAEH=DABCEH=ABCDEH=R

=>{AD} is the super key

Now E->C, D->AEH => ED->CEH => ED^+=EDCAEH=ABCDEH=R and E^+=EC≠R, D^+=R =>{ ED} is a super key.

$(ABH)^+$=ABCDEH=R, $(DH)^+$=ABCDEH =R =>{ABH} and {DH} are super keys.

A Key Finding Algorithm:

1 .K := R (*K is initialized as a super key*)

2. for each *A* in K do

3.{ if $(K - A)^+_F := R$ then K := K− *A }*

(F is the set of functional dependencies and* $(K - A)^+_F$ *is the closure of (K-A) with respect to F*)*

Example1: If R(A,B,C) and F={A->B,B->C}. Find the key of R

Solution: K=ABC

$(K-A)^+$ = $(BC)^+$ =BC (as B->C)

$(K-A)^+$ does not contain all attributes of R(A,B,C)

i.e.BC!=ABC =>K:=ABC

$(K-B)^+$=$(ABC-B)^+$=$(AC)^+$=ABC (as A->B and B->C)

$(K-B)^+$ contains all the attributes of R =>K:=K-B=AC

$(K-C)^+$=$(AB)^+$ =ABC(as A->B and B->C)

$(K-C)^+$ contains all the attributes of R =>K:=K-C=AC-C=A

So the key of relation R(A,B,C) ,with the given functional dependencies, is the key K={A} .

- *Find a key for each of the following questions:*
  - *R = {A, B, C }, F = {A →B, B →C, C →A }*
  - *R = {D, E, F }, F = {D →E, E →D, D →F }*
  - *R = {G, H, I }, F = {G →H, G →I }*
  - *R = {C, E, J }, F = {CE →J }*
  - *R = {C, E, G }, F = {}*
  - *R = {I, L }, F = {I →L }*
  - *R(A,B,C,D,E,F,G,H,I,J) ,F={AB->C,A->DE,B->F,F->GH,D->IJ}.*

Inferring Additional Keys: If *X = {A1,…, Ai,…, Ak } be a relation schema (R, F) key, where* $X \subseteq R$,

and $W \rightarrow Z \in F$ *(*$Z \not\subseteq W$ *,* $Z \subseteq X$ *and* $W \not\subseteq X$ *) then* $Y = (X - Z) \cup W$ *is also a relation schema (R, F) key,*

e.g. R = {A, B, C, D }, F = {AB $\rightarrow$*C, C* $\rightarrow$*D, D* $\rightarrow$*B }. X = AB is a key of (R, F) because* $(AB)^+ = ABCD = R$. *since D* $\rightarrow$*B* $\in F$, *Y = AD is another key of (R, F) because* $(AC)^+ = ABCD = R$.

since C $\rightarrow$*D* $\in F$, *Z = AC is a key of (R, F), as well.*

Covers:

Definition: F covers G if every FD in G can be inferred from F (i.e., if $G^+ \subseteq F^+$)

Equivalence of sets of functional dependencies:

Definition: F and G are equivalent if F covers G and G covers F.(i.e. $F^+ = G^+$).

Two sets of FDs F and G are equivalent if: -

every FD in F can be inferred from G, and every FD in G can be inferred from F.

Example: Show that F={ A->C, AC->D, E->AD, E->H} and G={ A->CD, E->AH} are equivalent or not? **Solution**: In F, we have A->C => AA->AC (by augmentation rule)=> A->AC (as AA=A).

A->AC, AC->D => A->D (by transitivity rule). Again A->C, A->D => **A->CD** (by union rule).

In F, we have E->AD => E->A and E->D (be decomposition rule) .

Now E->A, E->H =>**E->AH** (by union rule) . Thus G={ A->CD, E->AH} is inferred from F.

In G, we have A->CD => **A-> C** and A->D (by decomposition rule).

Now, we have A->D => **AC->D** (by augmentation rule). Again in G, E->AH => E->A, **E->H.**

We have E-> A, A->D=> E->D (by transitivity rule) . Again E->A,E->D => **E->AD** (by union rule).Thus F={ A->C, AC->D, E->AD, E->H} is inferred from G. Hence F and G are equivalent.

Non redundant cover:

Definition: If we have a set of FDs F, then we say that it is nonredundant if no proper subset F' of F is equivalent to F i.e.no F' exists such that $F'^+ = F^+$.

Non redundant cover algorithm:

Input: A set of FDs F.

Output: A non redundant cover of F.

```
G:=F; /* Initialize G to F*/
for each FD X->Y in G do
        if each FD X->Y in G do if X-> Y in G do
                if X->Y belongs to {F-(X->Y)}+ then F:= { F-(X-Y)};
G:=F /* G is the non redundant cover of F*/
```

Example: If F={ A->BC, CD->E, E->C, D->AEH, ABH-> BD, DH->BC}. Find the redundant and non redundant cover of F.

Solution: We have D-> AEH => D->E (by decomposition rule)

Now D->E => CD-> E (by augmentation rule)

So **CD->E is derived from D->AEH**.

Again D->AEH(given)=> D->A (by decomposition rule)

Now D->A, A->BC(given) => D->BC (by transitivity rule)

So D->BC=> DH->BC (by augmentation rule) .

Hence **DH->BC is derived from D->AEH and A->BC**.

Redundant FDs ={ CD->E, DH->BC}

Non redundant FDs F:={ F-(CD->E, DH->BC)}={ A->BC, E->C, D->AEH, ABH-> BD}

Extraneous attribute:

- An attribute A of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies.

- If F is a set of functional dependencies and the functional dependency X->Y in F .
  - Attribute A is extraneous in X if A∈ X , and
    F|=(F-{X->Y})U{(X-A)->Y}
  - Attribute A is extraneous in Y if A∈Y, and the set of functional dependencies
    (F-{X->Y})U{X->(Y-A)} |=F

Canonical cover F_C : Canonical cover F_C is a set of dependencies such that F logically implies all dependencies in F_C and F_C logically implies all dependencies in F. (F|= F_C and F_C |= F).

Properties of F_C:

- No functional dependency in F_C contains an extraneous attribute.
- Each left side of a functional dependency in F_C is unique.(i.e. there are no functional dependency in X1->Y1 and X2->Y2 in F_C such that X1=X2).

Algorithm of Canonical cover for functional dependencies (FDs)F:

repeat
Use the union rule to replace any fds in F of the form X1->Y1 and X1-> Y2 with X1->Y1Y2
Find a functional dependency X->Y with an extraneous attribute either in X or Y
If an extraneous attribute is found, delete it from X->Y
Until F doesn't change.

Problem: If R(A,B,C), F={ A->BC, B->C, A->B, AB->C} . Compute the canonical cover for F.

- Solution:
- There are two functional dependencies with the same set of atributs on the left side of the arrow A : A->BC,A->B and we combine these two FDs into A->BCB => A-> BC (as BB=B).
  A->BC => A->B, A->C (By decomposition rule)
- A is extraneous in AB->C since B->C =>AB->C (By augmentation rule).
- C is extraneous in A->BC since A->B, B->C => A->C (By transitivity rule)
  and A->B, A ->C => A->BC (By union rule)
- Thus our canonical cover is F_C ={ A->B, B->C}.

Minimal Sets of FDs: A set of FDs is minimal if it satisfies the following conditions:

(1) Every dependency in F has a single attribute for its RHS.

(2) We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.

(3) We cannot replace any dependency X -> A in F with a dependency Y -> A, where Y is proper-
subset-of X and still have a set of dependencies that is equivalent to F.

Minimal cover of a set of FDs: A minimal cover of a set of functional dependencies E is a minimal set of dependencies(in the standard canonical form and without redundancy) that is equivalent to E.

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets

Algorithm for finding a minimal cover F for a set of functional dependencies E

1. Set F:=E.
2. Replace each functional dependency X->{A1,A2,…An} in F by n functional dependencies
   X->A1, X->A2,…,X->An (i.e. each FD in F is simple)
3. for each functional dependency X->A in F
   for each attribute B that is an element of X
   if{{F-{X->A}}U{(X-{B})->A} is equivalent to F, then replace X-> A with (X-{B})->A in F.
4. For each remaining functional dependency X->A in F if {F-{X->A}} is equivalent to F, then remove X->A from F.

Problem: Find the minimal cover of the following functional dependencies:
F={ A->BC, CD->E, E->C, D->AEH, ABH-> BD, DH-> BC}

Solution: Convert all FDs into simple FDs by using decomposition rule(IR4):
A->BC=> A->B, A->C

D->AEH=>D->A, D->E, D->H
ABH->BD =>ABH->B, ABH->D
DH->BC=>DH->B, DH->C

So F={ A->B, A->C, CD->E, E->C, D->A, D->E, D->H, ABH->B, ABH->D, DH->B, DH->C}

But we have already proved that CD->E and DH->BC are redundant FDs. So remove them from F. Hence F={ A->B, A->C, E->C, D->A, D->E, D->H, ABH->B, ABH->D}.

Consider ABH->B and ABH->D.

A->B=>AH->B (By IR2: Augmentation rule)
=>ABH->BB(
=>ABH->B

=>ABH->B and it is redundant & can be derived from A->B and hence remove it from F.

. AH->D =>ABH->D (By augmentation rule) and it is redundant and can be derived from AH->D and hence remove it from F. So the minimal cover of F is given by
F= { A->B, A->C, E->C, D->A, D->E, D->H, AH->D}.

Functional dependencies and Keys:

Definition: If R(A1A2A3…An) be a relation schema, F be the set of functional dependencies, then a key 'K' of R is a subset of R such that K->A1A2A3…An $\in F^+$ and for any Y$\subseteq$ A1A2A3…An doesn't belongs to F^+.

Full functional dependency(FFD):

Definition : If X and Y are attributes of a relation R then Y is fully functionally dependent on X if Y is functionally dependent on X but not on any proper subset of X.

or

A FD X -> Y is FFD where removal of any attribute from X means the FD does not hold any more

i.e. Y is FFD on X if X->Y but Z ($\subseteq$ X) doesn't determine Y.

Example: 1) SUPPLIER

SNO	SNAME	STATUS	CITY
s_1	Suneet	20	Quadian
s_2	Ankit	10	Amritsar
s_3	Amit	30	Amritsar

(SNO,STATUS)->CITY but SNO->CITY , STATUS->CITY

{CITY} is FD on (SNO,STATUS) but {CITY} is not FFD on {SNO,STATUS}

because {CITY} is FD on {SNO} and {CITY}.

Example:2) SHIPMENT

SNO	PNO	QTY
S1	P1	270
S1	P2	300
S1	P3	700

S2	P1	270
S2	P2	700
S3	P2	300

{QTY} is FFD on (SNO,PNO) because

(i) (SNO,PNO)->QTY
(ii) (a) SNO doesn't determine QTY and (b)PNO doesn't determine QTY

Prime and nonprime attribute:

- A **Prime attribute** must be a member of *some candidate key/primary key.*
- A **Nonprime attribute** is not a prime attribute— that is, it is not a member of any candidate key.
- Example: R(ABCDEH), F={ A->BC, CD->E, E->C, AH->D}

=> $(AH)^+$ = AHD (as AH->D)
=ABCHD (as A->BC)
=ABCHDE(as CD->E)
=ABCDEH=R
=>AH->ABCDEH
=>{AH} is the candidate key of R
=> A and H are prime attributes whereas B,C,D,E and H are nonprime attributes.

Partial dependency:

Definition: If R be a relation schema with the functional dependencies F defined on the attributes of R and K as a candidate key, if X is a proper subset of K ($X \subseteq K$) and if F|=->A then A is said to be partially dependent on K.

i.e. Attribute Y is partially functionally dependent on X1X2(=K=key) if X1->Y or X2-> Y.

i.e. If Any attribute A is functionally dependent on part of the key(K1) (i.e.K1->A), then A is partially functionally dependent on the key.

i.e. A FD X -> Y is partial dependency where removal of any attribute from X means the FD holds any more.

Example: R=(A,B,C,D), F={AB->C, B->D}
(AB)+ = ABC (as AB->C)
= ABCD(as B->D)=R
AB is the key of R.
But given B->D and B $\subseteq$ AB(=key)
D is partially functionally dependent on the key AB.

Multi valued dependency(MVD):

Definition: If X, Y and Z are three attributes of a relation R such that for each value of X there is a set of values for Y (X->->Yi.e.X multi determines Y) and set of values for Z(X->->Zi.e. Y multi determines Z) then we say that X multi determines Y and Z denoted by X->->Y|Z or Y and Z are multi valued dependency on X, where the set of values for Y and Z are independent of each other.

Or

- A multi valued dependency X->>Y specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: if two tuples t1 and t2 exist in r such that t1[X]=t2[X], then two tuples t3 and t4 should also exist in r with the following properties:
    - t3[X]=t4[X]=t1[X]=t2[X]
    - t3[Y]=t1[Y] and t4[Y]=t2[Y]
    - t3[Z]=t2[Z] and t4[Z]=t1[Z]

Trivial and nontrivial MVD:

Definition: A MVD X->-> Y in R is called a trivial MVD if (a) Y⊆ X or (b) XUY=R.

A MVD X->-> Y in R is called a nontrivial MVD if neither(a)nor (b) is satisfied.

- A trivial MVD doesn't specify a constraint on a relation while a non trivial MVD does specify a constraint.
- There are two kinds of trivial MVDs:
  (a) X->-> Null set , where null set contains empty set of attributes.
  (b)X->->A-X, where A comprises all the attributes in a relation.
- Relations containing nontrivial MVDs tend to be all key relations i.e their key is all their attributes taken together.

Theory of MVD (or inference rules or FD and MVD): If X,Y,Z and W are subsets of relation

schema R={A1,A2,…,An}, then we have the followings rules for FD and MVD.

1) IR1-Reflexivity for FDs: Y⊆ X =>X->Y.
2) IR2-Augmentation for FDs: X->Y => XZ->YZ or XZ->Y
3) IR3-Transitivity for FDs: X->Y, Y->Z =>X->Z
4) IR4-Complementation for MVDs: X->->Y=>X->->(R-XUY)
5) IR5-Augmentaion for MVD:X->Y, Z⊆W =>WX->->YZ
6) IR6-Transitivity for MVDs: X->->Y, Y->->Z =>X->->(Z-Y)
7) IR7-Replication rule for FD to MVD: X->Y=>X->->Y
8) IR8-Coalescence rule for FDs and MVDs: If X->->Y and there exist W with the properties that
   (a)W∩Y= Null set,(b) W->Z, and (c)Y⊆ Z, then X->Z.
9) IR9-Reflexivity for MVDs: Y⊆ X =>X ->->Y or X->->X.
10) IR10-Intersection for MVDs: X->->Y, X->->Z=>X->->Y∩Z
11) IR11-Pseudo-transitivity for MVDs: X->->Y, YW->->Z=>XW->->(Z-WY)
12) IR12-Mixed(Pseudo) transitivity for MVDs: X->->Y, XY->->Z=>X->->(Z-Y)
13) IR13-Union rule for MVDs:X->->Y,X->->Z=>X->->YZ
14) IR14-Difference rule for MVDs: X->->Y, X->->Z =>X->->(Y-Z), X->->Z-Y
15) IR15-Decomposition/Projectivity rule for MVDs: X->->Y, X->->Z=>X->->(Y∩ Z),
    X->->(Y-Z)and X->->(Z - Y).

- If D denotes a set of FDs and MVDs , then the closure D^+ of D is the set of all FDs and MVDs logically implied by D.
- The list of inference rules for FDs and MVDs are sound and complete.
- Sound rules do not generate any dependencies that are are not logically implied by D.
- Complete rules allow us to generate all dependencies in D^+.

Question on MVD: If R=(A,B,C,G,H,I), D={A->->B, B->->HI, CG->->H}. Find the several members of closure of D(D^+).

Answer: (i)A->->B=>A->->R-B-A=CGHI (By complementation rule of MVD)
=> A->->CGHI.
(ii) A->->B, B->->HI =>A->->HI-B(By transitivity rule of MVD)

=>A->->HI. (HI-B=HI)

(iii) B->->HI , H⊆HI and CG->H and CG ∩HI =null set => B->H

(By Coalescence rule of MVD)

(iv)A->->CGHI, A->>HI =>CGHI-HI=CG i.e. A->CG (By difference rule of MVD)

Universal relation schema(R): The universal relational schema R={A1,A2,…,An} includes all the attributes of the database and every attribute name is unique.

Decomposition: Decomposition is the process of splitting a relation into its projections that will not be disjoint. The decomposition (D) of a universal relation schema R={A1,A2,A3,..An) is its replacement by a set of relation schema s D={ R1, R2,…,Rn} such that Ri ⊆ R(1≤ i≤m) and R1UR2UR3U…=R(**attribute preservation** condition of decomposition).

- Decomposition helps in eliminating some of the problems of bad design such as redundancy, inconsistencies and anomalies.
- When required , the DBA decides to decompose an initial set of relation schemes.

Desirable properties of decomposition:

- Attribute preservation
- Lossless-join decomposition
- Dependency preservation
- Lack of redundancy

Types of decomposition:

- Lossy (or Lossy join) decomposition
- Lossless(or non additive or non loss) join decomposition

Lossy(or Lossy join) decomposition:

- It is a property of decomposition , which ensures that spurious(extra) tuples are generated when relations are reunited through a natural join operation.
- The decomposition R(A,B,C) into R1 and R2 is lossy when the natural join of R1 and R2 does not yield the same relation as in R but yield spurious(extra) tuple(s).
- **Example**: In table1 i.e. R(ABC), A->B, C->B. R(ABC) is decomposed to R1(AB) and R2(BC) and then R3(ABC) is the natural join of R1(AB) and R2(BC) and in R3(ABC) neither B->A nor B->C is true. Also R3(ABC) ≠ R(ABC) because R3 contains some spurious(extra) tuples and so the decomposition of R(ABC) into R1(AB) and R2(BC) is a lossy decomposition .

R: ABC		
A	B	C
A1	B1	C1
A3	B1	C1
A2	B2	C3

A4	B2	C4

R1:AB	
A	B
A1	B1
A3	B1
A2	B2
A4	B2

R2:BC	
B	C
B1	C1
B1	C2
B2	C3
B2	C4

R3:ABC		
A	B	C
A1	B1	C1
A1	B1	C2
A3	B1	C1
A3	B1	C2
A2	B2	C4
A2	B2	C3
A4	B2	C3
A4	B2	C4

Lossless (or non additive or non loss) join decomposition:

- The word loss in lossless refers to loss of information.
- It is a property of decomposition which ensures that no spurious tuples are generated when relations are reunited through a natural join operation.
- The decomposition of R(X,Y,Z) into R1(X,Y) and R2(X,Z) is lossless if for attributes X, common to both R1 and R2, either X->Y or X->Z.
- A decomposition D={R1,R2,R3,…,Rm} of a relation R has lossless(non additive) join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds: natural join of the projections of all the relations in D from R =R.
- Example: Relation R(ABC) decomposed into R1(AB) and R2(BC) is lossless because the natural join of R1 and R2 = R3=R

R: ABC		
A	B	C
A1	B1	C1
A2	B2	C2
A3	B2	C1

A4	B1	C2

R1:AB	
A	B
A1	B1
A2	B2
A3	B2
A4	B1

R2:BC	
B	C
B1	C1
B2	C2
B3	C1
B4	C2

R3:ABC		
A	B	C
A1	B1	C1
A2	B2	C2
A3	B2	C1
A4	B1	C2

Module 3 (14 hrs)

Transactions :

- A transaction is a *logical unit* of program execution
- It is a combination of database updates which have to be performed together
- It is a logical unit of work.
- It is a unit of work with respect to concurrency and recovery.
- It is a sequence of operations including database operations that is atomic with respect to concurrency and recovery.
- An atomic execution unit that, when applied to a consistent database, generates a consistent but possibly different database.
- A short sequence of operations with the database which represents one meaningful activity in the user's environment

Transaction Processing Systems(TPS):

- Transaction processing systems(TPS) are the systems with large databases and hundreds of concurrent users that are executing database transactions
- Examples: Systems for reservations, banking , credit card processing, stock markets, super markets , super market checkout
- Typical OLTP Environments
    - Airline/ Railway Reservation Systems
    - Banking Systems (ATM)
    - Trading and Brokerage Systems
    - Hotel / Hospital Systems
    - Standard Commercial Systems

Properties (or ACID properties) of Transactions

- **Atomicity**: Either all updates are performed or none

- **Consistency**: If the database state at the start of a transaction is consistent, it will be consistent at the end of the transaction
- **Isolation**: When multiple transactions are executed concurrently, the net effect is as though each transaction has executed in isolation
- **Durability**: After a transaction completes (commits), its changes are persistent

States of transaction

- **Active**: Initial state; when the transaction is executing
- **Partially Committed**: When the last statement has finished execution
- **Failed**: On discovery that normal execution can no longer proceed

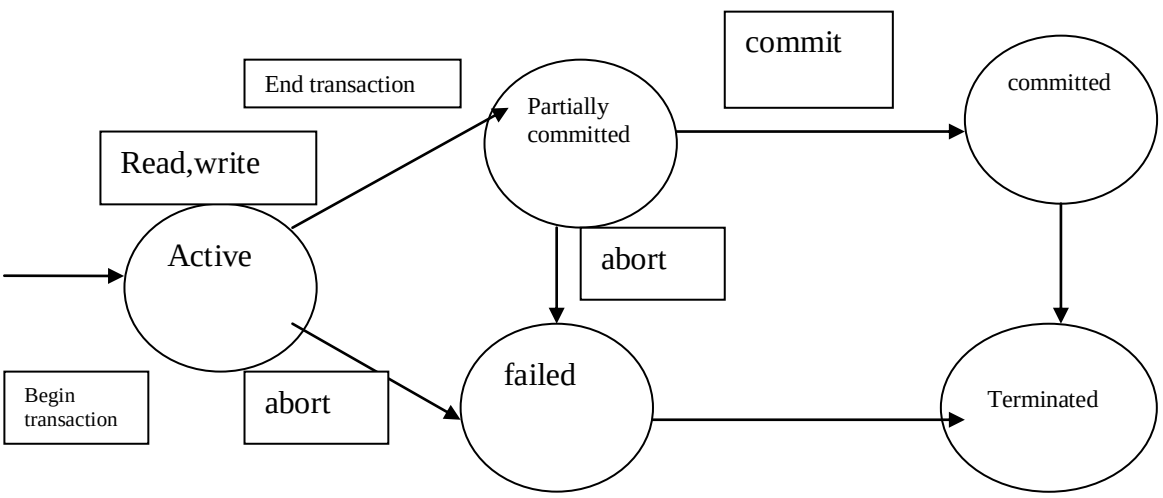
- **Aborted**: After the rollback is performed from a failed transaction
- **Committed**: After successful completion
- **Terminated**: Either committed or aborted

Database model: A database is basically represented as a collection of named data items. The size of data item is called the granularity and it can be a field of some record in the database ,or it may be a larger unit such as a record or even a whole disk block. The basic unit of data transfer from disk to main memory is one block.

Basic database access operations(processes) of a transaction:

Read operation: read_item(X): It reads a database item named X into a program variable named X.

Write operation: write_item(X): It writes the value of program variable X into the database item named X.



States of transaction

System log(DBMS journal): It keeps track of all transaction operations that affect the values of database items in order to be able to recover from failures that affect transactions

Log records: Log records are the types of entries that are written to the log and the action each performs.

In these entries, T refers to a unique transaction_id that is generated automatically by the system and is used to identify each transaction:

1.[start-transaction, T] : Indicates that transaction T has started execution.

2.[write_item, T,X,old_value,new_value]: Indicates that transaction T has changed the value of
database item X from an old value to new value.
3.[read_item, T,X]: Indicates that transaction T has read the value of database item X
4.[commit,T]: Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database , it is also known as commit point of a transaction
5.[abort,T]: Indicates that transaction T has been aborted

Concurrency control

- The process of managing simultaneous operations on the database without having them interfere with one another is called concurrency control
- Example: In airline reservation system, where the database containing available seats may accessed by many agents throughout the world
- The need for concurrency control (Why concurrent control is needed?) (Problem of concurrent transactions)
  - The lost update problem
  - The uncommitted dependency or dirty read/Temporary update problem
  - The inconsistent/incorrect summary problem
  - Unrepeatable read

The lost update problem(WW conflicts)

- This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect
- An apparently successfully completed update operation by one user can be overridden by another user
- Example: Let transactionsT1 and T2 are concurrently executing, T1 is withdrawing $10 from an account with balance balx, initially $100, and T2 is depositing $100 into the same account. (i.e.Balance=100-10+100=190)

Time	T1	T2	balx
t1		begin_transaction	100
t2	begin_transaction	read(balx)	100
t3	read(balx)	balx=balx+100	100
t4	balx=balx-10	write(balx)	200
t5	write(balx)	commit	90
t6	commit		90

- The loss of T2's update is avoided by preventing T1 from reading the value of balx until after T2' s update has been completed
- T1 overwriting the previous updates(i.e. 200) to write $90 at t5 therby 'losing' the $100 previously added to the balance

Dirty read (WR conflict)

(The uncommitted dependency/temporary update problem)

- This problem occurs when one transaction is allowed to see the intermediate results of another transaction before it has committed

Time	T3	T4	balx
t1		begin_transaction	100
t2		read(balx)	100
t3		balx=balx+100	100
t4	begin_transaction	write(balx)	200
t5	read(balx)	. . .	200

t6	balx=balx-10	rollback	100
t7	write(balx)		190
t8	commit		190

- The value of balx read by T3 at time t5(i.e. $200) is called dirty data, giving rise to alternative name, the dirty read problem ,but actually balx should be 90 instead of 190

The inconsistent analysis (incorrect summary) problem

- This problem occurs when a transaction reads several values from a database but a second transaction updates some of them during the execution of first

Time	T5	T6	balx	baly	balz	sum
t1		begin_transaction	100	50	25	
t2	begin_transaction	sum=0	100	50	25	0
t3	read(balx)	read(balx)	100	50	25	0
t4	balx=balx-10	sum=sum+balx	100	50	25	100
t5	write(balx)	read(baly)	90	50	25	100
t6	read(balz)	sum=sum+baly	90	50	25	150
t7	balz=balz+10		90	50	25	150
t8	write(balz)		90	50	35	150
t9	commit	read(balz)	90	50	35	150
t10		sum=sum+balz	90	50	35	185
t11		commit	90	50	35	185

- Sum should be =175 but it came to be =185, which is inconsistent result

Un(non) repeatable (fuzzy) Reads(RW conflicts)

- This problem occurs when a transaction T1 reads an item twice and the item is changed by another transaction T2 between the two reads
- Thus T1 receives two different values for the same data item
- Example:

T1	**T2**	**X=2000**	
		T1	**T2**
READ X		**2000**	
	UPDATE X		**3000**
READ X		**3000**	

Phantom read

- If a transaction T executes a query that retrieves a set of tuples from a relation satisfying a certain predicate, re-executes the query at a later time but finds that the retrieved set contains an additional (phantom) tuple that has been inserted by another transaction in the mean time
- This is some times referred to as phantom read .

Schedule

- A schedule S is defined as the sequential ordering of the operations of ‘n’ interleaved transactions
- A schedule is the chronological order in which instructions are executed in the system
- A schedule for a set of transactions must consist of all instructions of those transactions, and must preserve the order in which the instructions appear in each individual transaction

Serial Schedule

- A schedule S of n transactions is said to be a serial schedule if the operations of each transactions are executed consecutively without any interleaved operations from other transactions
- Each serial schedule consists of a sequence of a sequence of instructions from various transactions, where the instructions belonging to one single transaction appear together in that schedule.
- For a set of n transactions , there exist n! different serial schedules
- When several transactions are executed concurrently, the corresponding schedule no longer needs to be serial
- If two transactions are running concurrently, the operating system may execute one transaction for a little while, then perform a context switch, execute the second transaction for some time, and then switch back to the to the first transaction for some time, and so on.
- With multiple transactions , the CPU time is shared among all the transactions

Non-serial schedule

- A non-serial schedule is a schedule where the operations from a set of concurrent transactions are interleaved

Conflict operations

- WR conflict: T2 reads a data object previously written by T1
- RW conflict: T2 writes a data object previously read by T1
- WW conflict: T2 writes a data object previously written by T1

Complete schedule

- A schedule that contains either an abort or a commit as the last operation for each transaction whose actions are listed in it is called a complete schedule
- A complete schedule must contain all the actions of every transaction that appears in it.
- For any two conflicting operations(WR,RW,WW performed by Ti and Tj in order) one of the two must occur before the other in the schedule(Total order)

Serializability (Serialisable schedules)

- Any schedule that produces the same results as a serial schedule is called serial sable schedule
- On executing a set of concurrent transactions on a database the net effect should be as
  though the transactions were executed in *some* serial order.
- Serialisability theory attempts to determine the correctness of the schedules
- The rule of this theory is: A schedule S of n transactions is serial sable if it is equivalent
  to some serial schedule of the same ‘n’ transactions

Example:

Transaction T1:
read (A);
A = A – 50;
write (A);
read (B)
B = B + 50;
write (B);

Transaction T2:
read (A);
t = A * 0.1;
A = A – t;
write (A);
read (B) ;
B = B + t;
write (B);

Serial Schedules (Schedule-1)

(Take: A=1000, B=2000)

T1		
read (A)	A=1000	
A = A – 50;	A=950	Equivalent to T1 followed by T2
write (A)	A=950	
read (B)	B=2000	
B = B + 50;	B=2050	
write (B)	B=2050	

T2		
read (A)	A=950	
t = A * 0.1;	t=95	
A = A – t;	A=855	
write (A)	A=855	
read (B)	B=2050	
B = B + t;	B=2145	
write (B)		B=2145

A+B=855+2145=3000=>consistent state

Serial Schedules (Schedule-2)

T2		
read (A)	1000	
t = A * 0.1;	100	Equivalent to T2 followed by T1
A = A – t;	900	
write (A)	900	
read (B)	2000	
B = B + t;	2100	
write (B)	2100	

T1		
read (A) ;	900	
A = A – 50;	850	
write (A);	850	
read (B);	2100	
B = B + 50;	2150	Consistent state as
write (B) ;		2150

A+B=850+2150=3000

Schedule-3:Concurrent executions of serial schedule 1 and 2, producing consistent state as A+B=855+2145=3000

read(A)	1000		
A:=A-50	950		
write(A)	950		
		read(A)	950
		t:=A*0.1;	95
Context switch or interleaved in concurrent transactions T1 and T2 interleaved in concurrent transactions T1 and T2		A:=A-t;	855
		write(A)	855
read(B)	2000		
B:=B+50;	2050		
Write(B)	2050		
		read(B)	2050
		B:=B+t;	2145
		write(B)	2145

Concurrent transactions but inconsistent state

- **Not all concurrent executions result in a correct state**

- Example:Schedule1 and 2 in concurrent transactions T1 and T2

T1		T2	
read(A)	1000	read(A)	1000
A:=A-50;	950	t:=A*0.1;	100
		A:=A-t;	900
		write(A)	900
		read(B)	2000
write(A)	950		
read(B)	2000		
B:=B=50;	2050		
write(B)	2050		
		B:=B+t;	2100
		write(B);	2100

Here A+B=950+2100=3050≠3000 => The schedule-1 and 2 are in inconsistent state.

Serial schedule

- A serial schedule is a schedule in which either transaction T1 is completely done before T2 or transaction T2 is completed done before T1

Non serial schedule

- A non serial schedule is a schedule where the operations form a set of concurrent transactions are interleaved

Resultant equivalent

- Two schedules are called resultant equivalent if they produce the same final state of the database

Conflict Serializability

Let instructions I and J belonging to transactions T1 and T2 be executed consecutively by the DBMS.

1. I and J can be swapped in their execution order if I and J refer to different data elements
2. I and J can be swapped in their execution order **iff** I and J refer to the same data element and both perform a **read** operation only.
3. I and J are said to *conflict* if I and J belong to different transactions and at least one of them is a write operation.

Conflict Serializability and Conflict equivalent

- If a schedule S can be transformed to another S' by swapping non conflicting instructions, then S and S' are said to be *conflict equivalent*.
- A schedule S is said to be *conflict serializable* if it is conflict equivalent to some serial schedule S'.

Example(1) of Conflict equivalent

T1 T2 T1 T2

R(A) R(A)
W(A)
R(B)
R(B) EQUIVALENT TO W(A)
W(B)

W(B)

(W(A) and R(B) are non conflict and swapped) (R(B) and W(A) are conflict and swapped)

EQUIVALENT TO

T1 T2
R(A)
W(A)
R(B)
W(B)

Example(2) of Conflict equivalent

Schedule S

T1	T2
A=A+100	
B=B-100	
	A=A-7.06
	B=B*7.06

Schedule S'

T1	T2

A=A+100	
	A=A*7.06
B=B-100	
	B=B*7.06

Schedule S is conflict serializable since it it is conflict equivalent to the serial schedule S'

Two schedules produce the same outcome , but that are not conflict equivalent

T1 T2
R(A)
A:=A-50
W(A)
R(B)
B:=B-10
W(B)
R(B)
B:=B+50
W(B)
R(A)
A:=A+10
W(A)

The computation of T1 andT2 in the fig. is same as the computation of serial schedule<T1,T2> but are not conflict equivalent, because write(B) instruction of T2 conflicts with the read(B) instruction of T1

Example of a non-conflict serializable schedule

- T3 T4
  read(Q)
  write(Q)
  write(Q)

  This schedule is not conflict- serializable , since it is not equivalent to either the serial schedule<T3,T4> or <T4,T3>.

Serialisability test

Algorithm to determine wheather a schedule is serialisable or not?

- The steps of constructing a precedence graph are:
  - 1.Create a node for every transaction in the schedule
  - 2.Find the precedence relationships in conflicting operations.Conflicting operations are read-write(WR) or write-read(WR) or write-write(WW) on the same data item in two different transactions
    - (2.1)For a transaction Ti which reads an item A, find a transaction Tj that writes A later in the schedule.If such a transaction is found, draw an edge from Ti to Tj
    - (2.2)For a transaction Ti which has written an item A, find a transaction Tj later in the schedule that reads A. If such a transaction ids found, draw an edge from Ti to Tj
    - (2.3)For a transaction Ti which has written an item A, find a transaction Tj that writes A later than Ti. If such a transaction is found, draw an edge from Ti to Tj

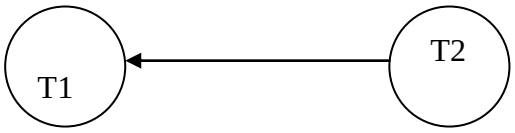
– 3.If there is any cycle in the graph, the schedule is not serialisable,otherwise, find the equivalent serial schedule of the transaction by traversing the transaction nodes with the node that has input edge
– Time taken for this test is of order of n^2 =O(n^2), where n is the number of transactions

Example-1 to illustrate serialisability test

An interleaved Schedule

Schedule	T1	T2
Read X	Read X	
Subtract 100	Subtract 100	
Read X		**Read X**
Write X	**Write X**	
Read Y		**Read Y**
Read y	Read Y	
Add 100	Add 100	
Display x+y		Display x+y
Write Y	**Write Y**	

- As per step1 of the algorithm we draw the two nodes for T1 and T2
- Transaction T2 reads data item X, which is subsequently written by T1, thus there is an edge from T2 to T1(clause 2.1)
- Also T2 reads data items Y, which is subsequently written by T1, thus there is an edge from T2 to T1(clause 2.1)
- However , that edge already exists, so we do not need to redo it.
- There are no cycles in the graph => The given schedule (Previous page) is serialisable
- The equivalent serial schedule(as per step 3) would be T2 followed by T1(the diagram is given in the next slide)
- The schedule is T2-T1
- Precedence graph for the schedule**: Not a cycle**:T2-T1=> The schedule is **serial sable**

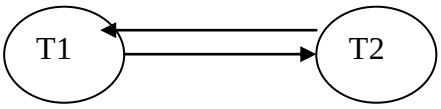


Example-2 to illustrate serialisability test

Schedule	Transaction 1 (T1)	Transaction 2 (T2)	Example values
Read X	Read X		X=50000

Subtract 100	Subtract 100		49900
Write X	**Write X**		X=49900
Read X		**Read X**	X=49900
Read Y		**Read Y**	Y=100000
Read X+Y		Display X+Y	149900
Read Y	Read Y		Y=100000
Add 100	Add 100		100100
Write Y	**Write Y**		Y=100100

- Schedule in example-2 is not serialisable , because in that schedule , the two edges that exist between nodes T1 and T2 are:
  - T1 writes X which is later read by T2(clause2.2), so there exists an edge from T1 to T2
  - T2 reads Y which is later written by T1(clause2.1), so there exists an edge from T2 to T1
- Thus the graph for the schedule will be : **Cycle**:T1-T2-T1=> Schedule is **not serialisable**



View Equivalent

If S is a schedule, then S' is a *view equivalent* schedule if

- For each data item Q, if a transaction Ti reads the initial value of Q in S, then it should read the initial value in S' also
- In schedule S, for each data item Q, if write(Q) of Tj precedes read(Q) of Ti, it should be the same in S'
- The same transaction that performs the final write(Q) in S, should perform it in S'.

View serializable

- A schedule S is view serializable if it is conflict equivalent to a serial schedule
- Every conflict-serializable schedule is view serializable, but there are view serializable schedules that are not conflict serializable
- The time taken for testing the view serialiability =$O(2^n)$

Example of view serializable:

T1	T2	T3
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

A view equivalent schedule:

T1: Read(Q)
T2: Write(Q)
T1: Write(Q)
T3: Write(Q)

The above Schedule is a view serialisable

Example of View equivalent Schedule-1

```
   T1
read (A)
A = A – 50;
write (A)
 read (B)
 B = B + 50;
 write (B)
```

```
                T2
            read (A)
          t = A * 0.1;
           A = A – t;
            write (A)
            read (B)
            B = B + t;
            write (B)
```

Schedule2

```
     T2
 read (A)
t = A * 0.1;
A = A – t;
write (A)
read (B)
B = B + t;
write (B)
```

Schedule1 is not view equivalent to Schedule2, since , in schedule 1, the value of account A read by transaction T2 was produced by T1, where as this case doesn't hold in schedule2

```
                T1
            read (A)
          A = A – 50;
            write (A)
            read (B)
            B = B + 50;
            write (B)
```

Constrained write assumption condition

- The definitions of conflict and view serializability are similar if a condition known as the constrained write assumption holds on all transactions in the schedule
- The constraint write assumption condition is as follows:
- Any write operation in wi(x) in Ti is preceded by a ri(x) in Ti and that the value written by wi(x) in Ti depends only on the value of x read by ri(x)

Blind write

The definitions of view serializability is less restrictive than that of conflict serializability under the unconstrained write assumption , where the value written by an operation Wi(X) in Ti can be independent of its old value from the database. This is called a blind rule.

Example:

```
   T1          T2          T3
 read(Q)
             write(Q)
```

write(Q)

write(Q)

- Here transactions T2 and T3 perform write(Q) operations without having performed a read(Q) operation
- Writes of this sort are called blind writes
- Blind writes appear in any view-serializable schedule that is not conflict serializable
- Every conflict serializable schedule is also view serializable; however (converse is not true)some view serializable schedules are not conflict serializable
- The problem of testing for view serializability has been shown to be NP-hard
- A schedule that is view serializable but not conflict serializable is characterized by *blind writes*.

Recoverability

- If a transaction Ti fails, for whatever reason, we need to undo the effect of this transaction to ensure the atomicity property of the transaction
- If a system that allows concurrent execution, it is necessary also to ensure that any transaction Tj that is dependent on Ti(i.e. Tj has read data written by Ti) is also aborted
  - Recoverble schedules
  - Cascadeless schedules
- **Recoverble schedules** :A recoverable schedule is one where, for each pair of transactions Ti and Tj such that Tj reads a data item previously written by Ti, the commit operation of Ti appears before the commit operation of Tj
  - Database systems require recoverable schedules.
  - Consider the following schedule

T1	T2
read(A)	
write(A)	
	read(A)
read(B)	

    - Suppose T2commits before T1
    - If T1fails before it commits then T2also has to be aborted.
    - However, T2is already committed.
    - A situation where it is impossible to recover from the failure of T1
    - This is an example of non recoverable schedule
- **Cascade less Schedule**: A cascade less schedule is one where, for each pair of transactions Ti and Tj such that Tj reads a data item previously written by Ti, the commit operation of Ti appears before the read operation of Tj
- The phenomenon , in which a single transaction failure leads to a series of transaction rollbacks, is called cascading rollback
- Cascading rollback is undesirable- leads to undoing a lot of work
- Restrict the schedules to those where cascading rollbacks do not occur.
- It is easy to verify that every cascade less schedule is also recoverable
- Even if a schedule is recoverable, to recover from the failure of a transaction, there is a need to rollback several transactions.

T	T1	T2
read(A)		
read(B)		
write(A)		

read(A)
write(A)
read(A)

- if T fails, then it will lead to rolling back T1 and T2.
- This is an example of cascading rollback.

Serializability Test

- It is just a test whether a given interleaved schedule is ok or has a concurrency related problem.
- However, it does not ensure that interleaved concurrent transactions do not have any concurrency related problems.
- This can be done using locks.
- Locking ensures serializability of executing transactions.

Problem of concurrent transactions

- The lost update problem
- The uncommitted dependency or dirty read/Temporary update problem
- The inconsistent/incorrect summary problem
- Unrepeatable read

Concurrency control techniques (algorithms)

- Conservative(or Pessimistic) approach
  - Locking
  - Timestamp
- Optimistic approach

Conservative (or Pessimistic) approach

- This approach causes transactions to be delayed in case they conflict with each other at some time in the future
- Pessimistic execution: If there is a validation according to compatibility of lock then only read, compute and write operations are performed
- The execution sequence is:
  Validate Read Compute Write
- Two pessimistic approaches are:
  - Locking
  - Time stamping

Optimistic approach

- These are based on the premise that conflict is rare so they allow transactions to proceed unsynchronized and only check for conflicts at the end, when transaction commits
- Optimistic execution is :
  Read Compute Validate Write

Types of Conservative/ Pessimistic approach

- Locking/2PL protocol
- Timestamp
  - Timestamp ordering protocol
  - Multi-version concurrency control

Lock-Based Protocols

- A lock/locking is a mechanism to control concurrent access to a data item
- A data can be locked by a transaction in order to prevent this data item from being accessed
- and updated by any other transaction

- The part of the database system which is responsible for locking or unlocking the data items is known as the lock manager
- Data items can be locked in two modes :

1. *exclusive (X) mode*. Data item can be both read as well as written. X-lock is requested using lock-X instruction. (Exclusive lock)

(i) It is requested by a transaction on a data item that it needs to update

(ii) No other transaction can place either a shared lock or an exclusive lock on a data item that has been locked in an exclusive mode

A data item locked in the exclusive mode can not be locked in the shared mode or in exclusive mode by any other transaction.

2. *shared (S) mode*. Data item can only be read. S-lock is requested using lock-S instruction. (Shared lock or read lock)

(i) It is requested by a transaction that wants to just read the value of data item

(ii) A shared lock on a data item does not allow an exclusive lock to be placed but permits any number of shared locks to be placed on that item.

A data item locked in shared lock cannot be locked in the exclusive mode by any other transaction but can be locked in the shared lock

- Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

Types of locks

- Binary lock
- Multiple –mode locks
  - Binary lock: This locking mechanism has two states for a data item; locked or unlocked.
  - Disadvantages of binary lock: It can't be used for practical purposes.
  - Multiple-mode lock: This locking mechanism has three states for to a data item: read locked or shared locked(read_lock(X)), write locked or exclusive locked. (write_lock(X) and unlocked (unlock(X))

Two Phase Locking(2PL) Protocol

- This is a protocol which ensures conflict-serializable schedules.
- **Definition of 2PL protocol**: A transaction follows the two-phase locking protocol if all locking operations precede the first unlock operation in the transaction.
- Phase 1: Growing Phase
  - transaction may obtain locks
  - transaction may not release locks
- Phase 2: Shrinking Phase
  - transaction may release locks
  - transaction may not obtain locks
- The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their lock points (i.e. the point where a transaction acquired its final lock).

- A transaction must acquire a lock on an item before on operating on the item.
- The lock may be read or write, depending on the type of access needed.
- Once the transaction releases a lock, it can never acquire any new lock.
- By using two phase locking protocol we can prevent the followings:
    - Preventing the lost update problem (WW conflict) using 2PL
    - Preventing the uncommitted dependency(or dirty read or temporary update) (WR conflict) problem using 2PL
    - Preventing the inconsistent analysis (incorrect summary) problem using 2PL
    - Preventing unrepeatable (RW conflict) read using 2PL

Preventing the problems occurred in concurrent transactions using two phase locking(2PL)

Preventing the lost update problem (WW conflict) using 2PL

Example: Let transactionsT1 and T2 are concurrently executing, T1 is withdrawing $10 from an account with balance balx, initially $100, and T2 is depositing $100 into the same account.(i.e.Balance=100-10+100=190)

Time	T1	T2	balx
t1		begin_transaction	100
t2	begin_transaction	write_lock(balx)	100
t3	write_lock(balx)	read(balx)	100
t4	wait	balx=balx+100	100
t5	wait	write(balx)	200
t6	wait	commit /unlock(balx)	200
t7	read(balx)		200
t8	balx=balx-10		200
t9	write(balx)		190
t10	commit/unlock(balx)		

Preventing the uncommited dependency(or dirty read or temporary update) (WR conflict)

problem using 2PL

Time	T3	T4	balx
t1		begin_transaction	100
t2		write_lock(balx)	100
t3		read(balx)	100
t4	begin_transaction	balx=balx+100	100
t5	write_lock(balx)	write(balx)	200
t6	wait	rollback/unlock(balx)	100
t7	read(balx)		100
t8	balx=balx-10		100
t9	write(balx)		90
t10	commit/unlock(balx)		90

Preventing the inconsistent analysis(incorrect summary) problem using 2PL

Time	T5	T6	balx	baly	balz	sum
t1		begin_transaction	100	50	25	
t2	begin_transaction	sum=0	100	50	25	0

t3	write_lock(balx)		100	50	250
t4	read(balx)	read_lock(balx)	100	50	250
t5	balx=balx-10	wait	100	50	250
t6	write(balx)	wait	90	50	250
t7	write_lock(balz)	wait	90	50	250
t8	read(balz)	wait	90	50	250
t9	balz=balz+10	wait	90	50	250
t10	write(balz)	wait	90	50	350
t11	commit/unlock(balx,balz)	wait	90	50	350
12		read(balx)	90	50	350
t13		sum=sum+balx	90	50	350
t14		read_lock(baly)	90	50	350
t15		read(baly)	90	50	350
t16		sum=sum+baly	90	50	350
t17		read_lock(balz)	90	50	350
t18		read(balz)	90	50	350
t19		sum=sum+balz	90	50	350
t20		commit/unlock(balx,baly,balz)	90	50	350

Preventing unrepeatable(RW conflict) read using 2PL

T1	T2	X=2000	
		T1	T2
READ X		2000	
	UPDATE X	Write_lock(X)	3000
READ X		2000	

Types of 2PL

- The Basic 2PL
- Strict 2PL

- Rigorous 2PL
- Conservative (Static) 2PL

The basic 2PL

- It allows release of lock at any time after all the locks have been acquired
- Disadvantages of the basic 2PL
  - Basic 2Pl suffers from the problem that it can result into loss of atomic/isolation property of transaction as theoretically speaking once a lock is released on a data item it can be modified by another transaction before the first transaction commits or aborts

Strict 2PL

- Most popular variation of 2PL
- It guarantees strict schedules and avoids the disadvantages of basic 2PL
- In strict 2PL, a transaction T doesn't release any of it's exclusive (write) locks until after it commits or aborts
- No other transaction can read or write an item that is written by T unless T has committed, leading to a strict schedule for recoverability
- Strict 2PL solves the problem of concurrency and atomicity
- Disadvantage is :

  It is not deadlock free i.e. it can introduce "Deadlock".

Rigorous 2PL

- It is a more common restrictive variation of strict 2PL.
- It guarantees strict schedules.
- In this variation, a transaction T doesn't release any of its locks (exclusive or shared) until after it commits or aborts .
- So it is easier to implement than strict 2PL .

Conservative (Static) 2PL

- It is a variation of 2Pl which requires a transaction to lock all the items it accesses before the

transaction begins execution, by pre declaring it's read_set and write_set.

Problems or Pitfalls with 2PL protocol

- Cascading rollback
- Deadlock
- Starvation

Cascading rollback

- It is a situation in which a single transaction leads to a series of rollbacks

Time	T1	T2	T3
t1	begin_transaction		
t2	write_lock(balx)		
t3	read(balx)		
t4	read_lock(baly)		
t5	read(baly)		
t6	balx=balx+baly		
t7	write(balx)		
t8	unlock(balx)	begin_transaction	
t9 …..		Write_lock(balx)	
t10 ….		read(balx)	
t11 …		balx=balx+100	
t12 …		write(balx)	

t13 …		unlock(balx)	
t14 ….		……	
t15	rollback	…….	
t16		……..	Begin_transaction
t17		…. ..	read_lock(balx)
t18		rollback	…….
t19			rollback

- Let us consider ,if transaction T1 fails after the read_lock(balx) operation of transaction T3.
- Then T1 must be rollback, which results into rollback of T2 and T3, so that actual value of
- balx from t1 will be accepted by T2 and T3.

Solutions to avoid cascading of rollbacks

- It can be done in two ways by using
  - Strict 2Pl protocol
  - Rigorous 2Pl protocol

Solution to avoid cascading of rollback by using strict 2PL protocol

- The strict 2PL protocol, requires ,that in addition to locking being 2 phase, all exclusive mode(lock_X(A)i.e.write_lock(A)) taken by a transaction must be hold until that transaction commits.
- This requirement ensures that any data written by an uncommitted transaction are locked in

exclusive mode until the transaction commits, preventing any other transaction from reading the data

Solution to avoid cascading of rollback by using rigorous 2PL protocol

- It requires that all locks to be held until the transaction commits
- It can be easily verified that, with rigorous 2PL, transactions can be serialized in the order in which

they commit.

Deadlock

- Definition: It is an impasse that may result when two (or more) transactions are each waiting for locks to be released that are held by the other.
- Deadlock occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T1 in the set.
- Hence each transaction in the set is waiting on awaiting queue , waiting for one of the other

transactions in the set to release the lock on an item.

Example:

Time	T1	T2
t1	BEGIN_TRANSACTION	
t2	WRITE_LOCK(BALX)	BEGIN_TRANSACTION
t3	READ(BALX)	WRITE_LOCK(BALY)
t4	BALX=BALX-10	READ(BALY)
t5	WRITE(BALX)	BALY=BALY+100
t6	WRITE_LOCK(BALY)	WRITE(BALY)
t7	WAIT	WRITE_LOCK(BALX)
t8	WAIT	WAIT

t9	WAIT	WAIT
t10	…	…
t11	…	…

General techniques for handling deadlocks

- Timeouts
- Deadlock prevention
- Deadlock detection and recovery

Time outs (or Lock timeouts)

- A transaction that requests a lock will wait for only a system-defined period of time.
- If the lock has not been granted within this period, the lock request times out .
- The transaction aborts and automatically restarts the transaction.
- This is used by several commercial RDBMS s

Deadlock Prevention Strategies

- Following schemes use transaction timestamps for the sake of deadlock prevention alone.
- Transaction timestamp (TS(T)): It is a unique identifier assigned to each transactions .A timestamp

is the system generated sequence number that is unique for each transaction. If transaction

T1 starts before transaction T2, then TS(T1)<TS(T2).T1 is called **older transaction** whereas T2 is

called **younger transaction.**

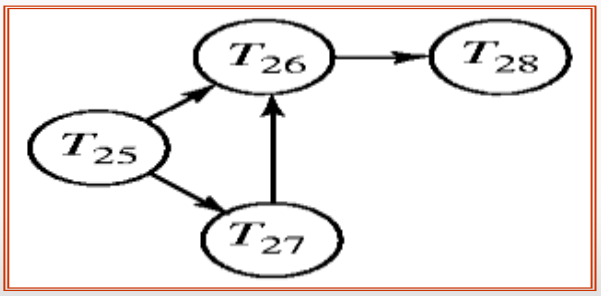
- There are two algorithms for dead lock prevention:
  - Wait-die and Wound-wait algorithms
  - No waiting(NW) and Cautious waiting(CW) algorithms
- **Wait-die** scheme — non-preemptive
  - older transaction may wait for younger one to release data item. Younger transactions never wait for older ones; they are rolled back instead.
  - A transaction may die several times before acquiring needed data item.
- **Wound-wait** scheme — preemptive
  - older transaction *wounds* (forces rollback) of younger transaction instead of waiting for it. younger transactions may wait for older ones.
  - May be fewer rollbacks than *wait-die* scheme.
- **No waiting (NW) algorithm**: If a transaction is unable to obtain a lock , it is immediately aborted and then restarted after a certain time delay without checking whether a deadlock will actually occur or not
- **Cautious waiting (CW) algorithm**: If Tj is not blocked (not waiting for some other locked item), then Ti is blocked and allowed to wait; otherwise abort Ti.

Deadlock Detection

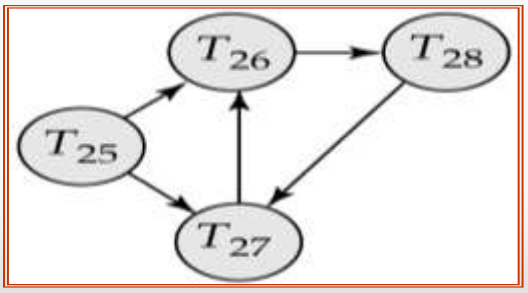
- **Deadlock detection algorithm**
  - Here the system checks if a state of deadlock actually exists.
  - Deadlock detection is usually handled by the construction of a wait-for graph(WFG) that shows the transaction dependencies.
  - Transaction Ti is dependent on Tj if transaction Tj holds the lock on a data item that Ti is waiting for.
  - The WFG is a directed graph with Pair $G = (V,E)$,
    - V is a set of vertices (all the transactions in the system)
    - E is a set of edges; each element is an ordered pair $Ti \rightarrow Tj$.

- If $Ti \rightarrow Tj$ is in E, then there is a directed edge from Ti to Tj, implying that Ti is waiting for Tj to release a data item.
- When Ti requests a data item currently being held by Tj, then the edge Ti Tj is inserted in the wait-for graph. This edge is removed only when Tj is no longer holding a data item needed by Ti.
- The system is in a deadlock state if and only if the wait-for graph has a cycle. Must invoke a deadlock-detection algorithm periodically to look for cycles.

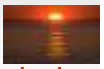
Deadlock Detection



Wait-for graph without a cycle



Wait-for graph with a cycle



- **Invoking the deadlock detection algorithm depends on two** factors:
  - How often does a deadlock occur?
  - How many transactions will be affected by the deadlock?
- **Frequency of deadlock detection**:
  - Since a cycle in the WFG is a necessary and sufficient condition for deadlock to exist, the deadlock detection algorithm generates the WFG at regular intervals and examines it for a cycle.
  - The choice of time interval between executions of the algorithm is important.
  - If the interval chosen is too small, deadlock detection will add considerable overhead.
  - If the interval is too large, deadlock may not be detected for a long period.
  - Alternatively a dynamic deadlock detection algorithm could start with an initial interval size, each time no deadlock is detected, the deadlock interval could be increased, for example, to twice the previous interval, and each time deadlock is detected, the interval could be reduced , e.g., to half the previous interval, subject to some upper and lower limits.

Deadlock Recovery

- When a detection algorithm determines that a deadlock exists, the system must recover from the deadlock
- When deadlock is detected :
  - Some transaction will have to rolled back (made a victim) to break deadlock. Select that transaction as victim that will incur minimum cost.
  - Rollback -- determine how far to roll back transaction
    - Total rollback: Abort the transaction and then restart it.

- More effective to roll back transaction only as far as necessary to break deadlock.

- Starvation happens if same transaction is always chosen as victim. Include the number of rollbacks in the cost factor to avoid starvation.
- The following are the several issues:
  - Choice of deadlock victim
  - How far to roll a transaction back
  - Avoiding starvation
- **Choice of deadlock victim:** Choosing which transaction to abort is known as victim selection. We should rollback those transactions that will incur the minimum cost. When deadlock is detected, the choice of which transaction to abort can be made using the following criteria:
  - The transaction which have the fewest locks
  - The transaction that has done the least work
  - The transaction that is farthest from completion
- **How far to roll back a transaction back:** Having decided to abort a particular transaction

,we have to decide how far to roll the transaction back. Clearly, undoing all the changes

made by a transaction is the simplest solution, although not necessarily the most efficient

It may be possible to resolve the deadlock by rolling back only part of the transaction.

- **Avoiding starvation**: Starvation occurs when a transaction can't proceed for an indefinite

period of time while other transactions in the system continue normally. This may occur

if the waiting scheme for locked items is unfair, giving priority to some transactions over

others. Starvation occurs when the same transaction is always chosen as the victim, and

the transaction can never complete. Starvation is very similar to live lock(left in a wait

state indefinitely, unable to acquire any new locks, although the DBMS is not in

deadlock), which occurs when the concurrency control protocol never selects a particular

transaction that is waiting for a lock.

- **Solutions to starvation:**
  - **1st Scheme :First Come First Served(FCFS) queue:** Transactions are enabled to lock an item in the order in which they originally requested the lock.
  - **2nd Scheme:** It allows some transactions to have priority over others but increases the priority of a transaction the no longer it waits, until it eventually gets the highest priority and proceeds.

- **Avoidance of starvation:**
  - The DBMS can avoid starvation by storing a count of the number of times a transaction has been selected as the victim and using a different selection criterion once this count reaches some upper point.
  - The wait-die and wound-wait schemes avoid starvation.

Database recovery:

- It is the process of restoring the database to a correct state in the event of a failure.
- A typical strategy for recovery may be summarized informally as follows:

(i)If there is extensive damage to a wide portion of the database due to catastrophic failures, such as disk crash , the recovery method restores a past copy of the database that was backup to archival storage(tape) and reconstructs a more current state by reapplying or redoing the operations of committed transactions from the backed up log, up to the time of failure

(ii) When the database is not physically damaged , but has become inconsistent due to non catastrophic failures, the strategy is to reverse any changes that caused the inconsistency by undoing some operations. It may be necessary to redo some operations in order to restore a consistent state of the database. In this case , we do not need a complete archival copy of the database. Rather , the entries kept in the online system log are consulted during recovery

Techniques for recovery from non catastrophic transaction failures

- Deferred update(No undo/redo algorithm)
- Immediate update(Undo/redo algorithm)
- Shadow paging
- ARIES recovery algorithm

Deferred update(No undo/redo algorithm)

- This technique does not physically update the database on disk until after a transaction reaches its commit point; then the updates are recorded in the database.
- Before reaching commit, all transaction updates are recorded in the local transaction workspace (or buffers).
- During the commit, the updates are first recorded persistently in the log and then written to the database.
- If a transaction fails before reaching its commit point, it will not have changed the database in any way, so UNDO is needed.
- It may be necessary to REDO the effect of the operations of a committed transaction from the log, because their effect may not yet have been recorded in the database

Immediate update(Undo/redo algorithm)

- In this technique , the database may be updated by some operations of a transaction before the transaction reaches its commit point.
- These operations are typically recorded in the log on disk by force writing before they are applied to the database , making recovery still possible.
- If a transaction fails after recording some changes in the database but before reaching its commit point, the effect of its operations on the database must be undone i.e. the transactions must be rolled back.

- In the general case of immediate update, both undo and redo may be required during recovery

Shadow paging (Lorie, 1977)

- It is an alternative to the log-based recovery schemes (Deferred and Immediate update)
- This scheme maintains two page tables during the life of a transaction:
  - a)A current page table
  - b)A shadow page table
- When the transaction starts, the two-page tables are the same.
- The shadow page table is never changed thereafter, and is used to restore the database in the event of a system failure.
- During the transaction, the current page table is used to record all updates to the database.
- When the transaction completes, the current page table becomes the shadow page table
- Advantages of shadow paging over the log based schemes
  - The overhead of maintaining the log file is eliminated, and recovery is significantly faster since there is no need for undo or redo operations
- Disadvantages
  - Data fragmentation
  - The need for periodic garbage collection to reclaim inaccessible blocks

ARIES recovery algorithm

- ARIES uses a steal/no-force approach for writing , and it is based on three concepts:
  - Write-ahead logging
  - Repeated history during redo
  - Logging changes during undo
- Write-ahead logging

The recovery mechanism must ensure that the BFIM(Before image-the old value of the data item before updating) of the data item is recorded in the appropriate log entry and that the log entry is flushed to disk before the BFIM is overwritten with the AFIM(after image-the new value after updating) in the database on disk.

- Repeated history during redo

The ARIES will retrace all actions of te database system prior to the crash to reconstruct the database state when the crash occurred. The transactions that were uncommitted at the time of the crash(active transactions) are undone.

- Logging during undo

It will prevent ARIES from repeating the completed undo operations if a failure occurs during recovery, which causes a restart of the recovery process.

ARIES recovery procedure: It consists of three main steps :

- Analysis
- REDO
- UNDO

Analysis step: This step identifies the dirty (updated) pages in the buffer and the set of transactions active at the time of the crash.

REDO phase: It actually reapplies updates from the log to the database. The REDO operation is applied only to the committed transaction.

UNDO phase: During this phase, the log is scanned backward and the operations of transactions that were active at the time of the crash are undone in reverse order.

The information's needed for ARIES to accomplish its recovery procedure includes:

- The log
- The transaction table
- The dirty page table
- Check pointing

The log : In ARIES, every log record has an associated log sequence number (LSN) that is monotonically increasing and indicate the address of the log record on disk. Each LSN corresponds to a specific change(action) of some transaction. Each data page will store the LSN of the latest log record corresponding to a change for that page. A log record is written for any of the following actions: updating a page(write), committing a transaction(commit), aborting a transaction (abort), undoing an update(undo), and ending a transaction(end).

Transaction table :It 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.

The dirty page table: It 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. The transaction table and dirty page table are needed for efficient recovery maintained by the transaction manager. When a crash occurs, these tables are rebuilt in the analysis phase of recovery.

Check pointing:

- It is the point of synchronization between the database and the transaction log file. All buffers are force-written to secondary storage.
- Check points are scheduled at predetermined intervals and involve the following operations:
- Writing all log records in main memory to secondary storage
- Writing a checkpoint record to the log file. This record contains the identifiers of all the transactions that are active at the point of check point
- Check pointing in ARIES consists of the following:
    - Writing a begin_checkpoint record to the log.
    - Writing an end_checkpoint record to log.
    - Writing the LSN of the begin_checkpoint record to a special file.
    - The special file is accessed during recovery to locate the last check point information.
    - With the end_checkpoint record, the contents of both the transaction table and dirty page table are appended to the end of the log. To reduce the cost, fuzzy check pointing is used so that the DBMS can continue to execute transactions during chekpointing

Database Security: It is the mechanism that protect the database against intentional or accidental threats.

Threat: It is a situation or event , whether intentional or accidental , that may adversely affect a system and consequently the organization.

Threats to databases: Threats to databases result in the loss or degradation of some or all of the following commonly accepted security goals: integrity, availability and confidentiality.

Loss of integrity: Integrity is lost if unauthorized changes are made to the data by either intentional or accidental acts. If the loss of system or data integrity is not corrected , continuous use of the contaminated system or corrupted data could result in inaccuracy, fraud , or erroneous decisions .**Database integrity** refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, modification, changing the status of data , and deletion.

Loss of availability: Database availability refers to making objects available to human user or a program to which they have a legitimate right.

Loss of confidentiality: Database confidentiality refers to the protection of data from unauthorized disclosure. Unauthorized , unanticipated , or unintentional disclosure could result in loss of public
confidence , embarrassment, or legal action against the organization.

Control measures: To protect database against all the thrats , it is common to implement four kinds of
control measures:

- Access control
- Inference control
- Flow control
- Encryption

Access control:

- The security mechanism of a DBMS must include provisions for restricting access to the database system as a whole. This function is called access control and is handled by creating user accounts and passwords to control the login process by the DBMS.
- The DBA is the central authority for managing a database system.
- The DBA's responsibilities include granting privileges to users who need to use the system and classifying users and data in accordance with the policy of the organization.
- The DBA has a DBA account in the DBMS, sometimes called a system or supervisor account, which provides powerful capabilities that are not made available to regular database accounts and users.
- DBA-privileged commands include commands for granting and revoking privileges to individual accounts, users, or user groups and for performing the following types of actions:
    - Account creation: This action creates a new account and password for a user or group of users to enable access to the DBMS.
    - Privilege granting: This action permits the DBA to grant certain privileges to certain accounts.
    - Privilege revocation: This action permits the DBA to revoke(cancel) certain privileges that were previously given to certain accounts.
    - Security level assignment: This action consists of assigning user accounts to the appropriate security classification level.
- The DBA is responsible for the overall security of the database system

Inference control

- Security for statistical databases must ensure that information about individuals can't be accessed.
- It is sometimes possible to deduce or infer certain facts concerning individuals from queries that involve only summary statistics on groups; consequently, this must not be permitted either.This problem, called **statistical database security.** The corresponding control measures are called inference control measures.
- The possibility of inferring individual information from statistical queries is reduced if no statistical queries are permitted whenever the number of tuples in the population specified by the selection condition falls below some threshold.
- Another technique for prohibiting retrieval of individual information is to prohibit sequence of queries that refer repeatedly to the same population of tuples.
- Another technique is partitioning the database. Partitioning implies that records are stored in groups of some minimum size; queries can refer to any complete group or set of groups, but never to subsets of records within a group.

Flow control

- It prevents information from flowing in such a way that it reaches unauthorized users.
- Channels that are pathways for information to flow implicitly in ways that violate the security policy of an organization are called **covert channels**

Encryption

- The encoding of the data by special algorithm that renders the data unreadable by any algorithm without the decryption key.
- Encryption can be be used to provide additional protection for sensitive portions of a database as well.
- An unauthorized user who accesses encoded data will have difficulty in deciphering it, but authorized users are given decoding or decryption algorithm(or keys) to decipher the data.

Authorization

- It is the granting of a right or privilege that enables a subject to have legitimate access to a system or a system's object

Authentication

- It is a mechanism that determines whether a user is who he or she claims to be.

Database security and authorization subsystem

- A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security of portions of a database against unauthorized access. It is now customary to refer to two types of database security mechanisms:
    - **Discretionary security mechanisms**: These are used to grant privileges to users, including the capability to access specific data files , records, or fields in a specific mode(such as read, insert, delete, or update).
    - **Mandatory security mechanisms**: These are used to enforce multilevel security by classifying the data and users into various security classes or levels and then implementing the appropriate security policy of the organization.

An Introduction to Data Mining

Why Data Mining:

- Credit ratings/targeted marketing:
    - Given a database of 100,000 names, which persons are the least likely to default on their credit cards?
    - Identify likely responders to sales promotions

- Fraud detection
  - Which types of transactions are likely to be fraudulent, given the demographics and transactional history of a particular customer?
- Customer relationship management:
  - Which of my customers are likely to be the most loyal, and which are most likely to leave for a competitor?
- Data Mining helps extract such information

Definition of data mining

- Data mining is the extraction or mining of knowledge from large amounts of data
  Example: Mining (extracting) of golds from rocks or sand is known as gold mining rather than rock or sand mining
- It is a step in the knowledge discovery process, which is the process of discovering interesting knowledge from large amounts of data stored either in databases, data ware houses, or other information repositories
- It is the process of semi-automatically analyzing large databases to find patterns that are:
  - valid: hold on new data with some certainty
  - novel: non-obvious to the system
  - useful: should be possible to act on the item
  - understandable: humans should be able to interpret the pattern
- Data mining is also known as Knowledge Discovery in Databases (KDD**)**

Definition of Knowledge Discovery in Databases (KDD): Knowledge Discovery in Data is the *non-trivial* process of identifying *valid*, *novel*, potentially *useful* and ultimately *understandable patterns* in data.

Knowledge discovery in databases (KDD): KDD is a process of consisting of an iterative sequence of the following steps:

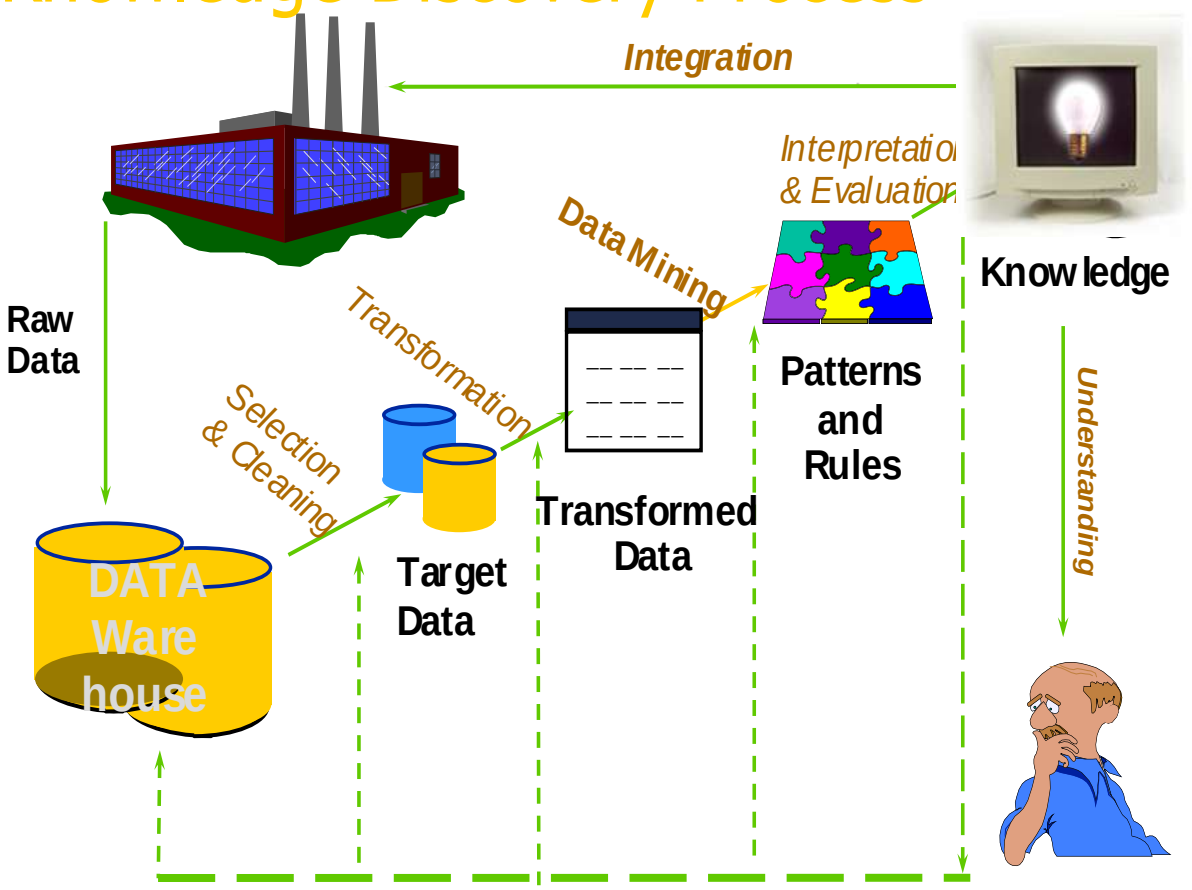
- Data cleaning: Noise and inconsistent data are removed.
- Data integration: Multiple data sources are combined.
- Data selection: Data relevant to the analysis task are retrieved from the database.
- Data transformation: Data are transformed or consolidated into forms appropriate for mining by performing summary or aggregation operations, for instance.
- Data mining: An essential process where intelligent methods are applied in order to extract data patterns.
- Pattern evaluation: Identifies the truly interesting patterns representing knowledge based on some interestingness measures.
- Knowledge presentation: Visualization and knowledge representation techniques are used to present the mined knowledge to the user.

:

Application Areas

Industry	Application
Finance	Credit Card Analysis
Insurance	Claims, Fraud Analysis
Telecommunication	Call record analysis
Transport	Logistics management
Consumer goods	promotion analysis
Data Service providers	Value added data
Utilities	Power usage analysis

Knowledge Discovery Process



Data Mining in Use

- The US Government uses Data Mining to track fraud
- A Supermarket becomes an information broker
- Basketball teams use it to track game strategy
- Cross Selling
- Target Marketing
- Holding on to Good Customers
- Weeding out Bad Customers

Data mining functionalities/techniques

- Concept/Class description
- Association analysis
- Classification and prediction
- Cluster analysis
- Outlier analysis
- Evolution analysis

Concept/Class description

Describes individual classes and concepts in summarized, concise, and yet precise term

Example: Classes of items for sale in all electronics store include Computers & Printers **Association analysis**

X=>Y is interpreted as "Database tuples that satisfy the conditions in X are also likely to

satisfy the conditions in Y"

Example: Contains (T, "Computer")=>contains (T,"S/W") where T=transaction

Age(X,"20…29") ^income(X,"20K…29K") =>buys (X, "CD player") where X=customer

Single and multidimensional association rule

Association rule that contains more than one attribute or predicate is called multidimensional

association rule. Association rule that contains a single attribute or predicate is called

single dimensional association rule.

Classification and prediction

Predicting the class level of data objects is known as classification, with known class levels of data objects(training data). Prediction is the data value prediction. Relevance analysis attempts to identify attributes that do not contribute to the classification or prediction process

Cluster analysis

- Analyzes data objects without consulting a known class label.
- The class levels are not present in the training data.
- Used to generate class labels.
- The data objects are clustered or grouped based on the principle of maximizing the intra class similarity and minimizing the interclass similarity.

Outlier analysis

- Outliers: are the data objects that do not comply with the general behavior or model of the data
- Most data mining methods discard outliers as noise or exceptions
- Outlier mining is the analysis of outlier data

Evolution analysis

- Data evolution analysis describes and models regularities or trends for objects whose behavior changes over time
- Distinct features of evolution analysis include time-series data analysis , sequence or periodicity pattern matching, and similarity-based data analysis

Data Warehouse: A data warehouse is a subject-oriented, integrated, time-varying, and non-volatile collection of data that is used primarily in organizational decision making.

Subject -oriented

- The warehouse is organized around the major subjects of the enterprise (such as customers, products, and sales) rather than the major application areas(such as customer invoicing, stock control, and product sales).
- This is reflected in the need to store decision-support data rather than application-oriented data

Integrated nature of data

- Integrated means bringing together data of multiple , dissimilar operational sources
- Identifies and defines the organizational key data items uniquely
- Identifies the logical relationships between them ensuring organization wide consistency in terms of:
  - Data naming and definition
  - Encoding structure
  - Measurement of variables

Time-variant (historical) nature of data

- Because data in the warehouse is only accurate and valid at some point in time or over some time interval
- Historical in nature
- Contains data that is date-stamped
- Operational data changes over a shorter time period

Non-volatile (Static nature) nature of data

- As the data is not updated in real time but refreshed from operational systems on a regular basis

- Data warehouse data is loaded on to the data warehouse database and
- Is subsequently scanned and used , but is not updated in the same classical sense as operational system's data which is updated through the transaction processing cycles

Characteristics of data ware houses:

- Multidimensional conceptual view
- Generic dimensionality
- Unlimited dimensions and aggregation levels
- Unrestricted cross-dimensional operations
- Dynamic spare matrix handling
- Client/Server architecture
- Multi-user support
- Accessibility
- Transparency
- Intuitive data manipulation
- Consistent reporting performance
- Flexible reporting

Advantages (benefits) of data ware house

- Potential high returns on investment
- Competitive advantage
- Increased productivity of corporate decision making

Disadvantages (limitations/ problems) of data ware house

- Underestimation of resources for data loading
- Hidden problems with source systems
- Increased end-user demands
- Data homogenizations
- High demand for resources
- Data ownership
- High maintenance
- Long-duration projects
- Complexity of integration

Typical applications of data ware house

- Online analytical processing(OLAP)
- Decision-support system(DSS)
- Data mining
    - **Online analytical processing(OLAP):** It is used to describe the analysis of complex data from the data ware house. In the hands of skilled knowledge workers, OLAP tools use distributed computing capabilities for analysis that require more storage and processing power than can be economically and efficiently located on an individual desktop.

      Or
    - It is the dynamic synthesis, analysis, and consolidation of large volumes of multi-dimensional data.
        - **Applications of OLAP**
            - Finance
            - Sales
            - Marketing
            - Manufacturing

- **Online transaction processing(OLTP):**It includes insertions, updates, and deletions, while also supporting information query requirements.
- **Decision support systems(DSS):** It is also known as Executive Information Systems(EIS) that support an organization's leading decision makers with higher level data for complex and important decisions .
- **Data mining:** It is used for knowledge discovery, the process of searching data for unanticipated new knowledge

- **OLAP Benefits:**
  - Increased productivity of business end-users, IT developers, and consequently the entire organization.
  - Reduced backlog of applications development for IT staff by making end-users self-sufficient enough to make their own schema changes and build their own models.
  - Retention of organizational control over the integrity of corporate data as OLAP applications are dependent on data warehouses and OLTP systems to refresh their source level data.
  - Reduced query drag and network traffic on OLTP systems or on the data ware house.
  - Improved potential revenue and profitability by enabling the organization to respond more quickly to market demands.

Database systems and the internet

- Many electronic commerce(e-commerce)and other internet applications provide web interfaces to access information stored in one or more databases. These databases are often referred to as **data sources.**
- It is common to use two-tier and three-tier client/server architectures for internet applications.
- **Internet**: It is the worldwide collection of interconnected computer networks.
- **Intranet**: It is a website or group of sites belonging to an organization, accessible only by the numbers of the organization.
- **Extranet**: It is an intranet that is potentially accessible to authorized outsiders.
- **E-Commerce**: Customers can place and pay for orders via the business web site.
- **E-business:** Complete integration of internet technology into the economic infrastructure of the business

Search engines

Search engine is a program that searches documents for specified keywords and returns a list of the documents where the keywords were found. Although search engine is really a general class of programs , the term is often used to specifically describe systems like Google, Alta Vista and Excite that enable users to search for documents on the world wide web and USENET newsgroups. Typically, a search engine works by sending out a spider to fetch as many documents as possible. Another program, called an indexer, then reads these documents and create an index based on the words contained in each document. Each search engine uses a proprietary algorithm to create its indices such that, ideally, only meaningful results are returned for each query.

Semi-structured data model:

- **Semi-structured data:** It is the data that may be irregular or incomplete and have a structure that may change rapidly or unpredictably.

- **Structured data:** The information stored in the database is known as structured data because it is represented in a strict format i.e. tabular.
- **Unstructured data:** There is a very limited indication of the type of data.
- **Example:** (1) A text document that contains information within it.
  (2)Web pages in HTML that contains some data

Data model for semi-structured data-Object Exchange Model(OEM)

- Semi-structured data may be displayed as directed graph.
- This model somewhat resembles the object model in its ability to represent complex objects and nested structures.
- The labels or tags on the directed edges represent the schema names: the name of attributes, object types(or entity types or classes), and relationships.
- The internal nodes represent individual objects or composite attributes.
- The leaf nodes represent actual data values of simple(atomic) attributes.
- The figure below illustrates this.

Extensible Markup Language(XML) and Web databases

- It is a meta-language (a language for describing other languages) that enables designer to create their own customized tags to provide functionality not available with HTML.
- XML is a restricted version of standard generalized markup language (SGML) designed especially for web documents.

Document type definitions (DTD): It defines the valid syntax of an XML document.

Advantages of XML

- Simplicity
- Open standard and platform/ vender- independent
- Extensibility
- Reuse
- Separation of content and presentation
- Improved load balancing
- Support for the integration of data from multiple sources
- Ability to describe data from a wide variety of applications
- More advanced search engines
- New opportunities

Object and Object relational databases

Object oriented data model(OODM): It is a logical data model that captures the semantics of objects supported in object oriented programming(OOP).

Object Oriented Database(OODB): It is a persistent and sharable collection of objects defined by an OODM.

Object Oriented DBMS(OODBMS): The manager of an OODB is called as OODBMS.

Object relational DBMS(ORDBMS): ORDBMS consciously try to add OODBMS features to an RDBMS.

Note: OODBMSs and ORDBMSs both support user-defined ADTs, structured types, object identity and reference types **,** and inheritance , both support a query language for manipulating collection types.

ORDBMSs support an extended form of SQL, and OODBMSs support ODL/OQL. Both support

concurrency and recovery.

Object orientation = Abstract data types(ADT) + Inheritance+ Object identity

OODBMS= Object orientation +Database capabilities

RDBMS traditional business applications:

- Order processing
- Inventory control
- Banking
- Airline reservation

Weakness of RDBMS: leads to development of OODBMS, which are:

- Poor representation of real world entities
- Semantic overloading
- Poor support for integrity and enterprise constraints
- Homogenous data structure.
- Limited operations
- Difficulty handling recursive queries
- Impedance mismatch
- Concurrency, schema changes, and poor navigational access

Advanced database applications: The following new applications can't be easily designed in RDBMS due

weaknesses of RDBMS :

- Computer Aided Design(CAD)
- Computer Aided Manufacturing (CAM)
- Computer Aided Software Engineering(CASE)
- Network management system
- Office information system and multimedia system
- Digital publishing
- Geographical information system(GIS)
- Interactive and dynamic web sites

Main concepts used in object oriented databases

- **Object identity**: Objects have unique identities that are independent of their attribute values.
- **Type constructors:** Complex object structures can be constructed by recursively applying a
  set of basic constructors, such as tuple, set, list, and bag.
- **Encapsulation of operations:** Both the object structures and the operations that can be applied
  to objects are included in the object class definitions.
- Programming language compatibility: Both persistent and transient objects are handled seamlessly.
  Objects are made persistent by being attached to a persistent collection or by explicit naming.
- **Type hierarchies and inheritance**: Object types can be specified by using a type hierarchy,
  which allows the inheritance of both attributes and methods of previously defined types. Multiple inheritance is allowed in some methods.
- **Extents**: All persistent objects of a particular type can be stored in an extent.
  Extents corresponding to a type hierarchy have set/subset constraints forced on them.

- **Support for complex objects**: Both structured and unstructured complex objects cn be stored and manipulated.

Advantages of OODBMS

- Enriched modeling capabilities
- Extensibility
- Removal of impedance mismatch
- More expressive query language
- Support for schema evolution
- Support for long duration transactions
- Applicability to advanced database applications
- Improved performance

Disadvantages of OODBMS

- Lack of universal data model
- Lack of experience
- Lack of standards
- Competition
- Query optimization compromises encapsulation
- Locking at object level may impact performance
- Complexity
- Lack of support for views and security

Advantages of ORDBMS

- Reuse and sharing
- Increased productivity
- Use of experience in developing RDBMS

Disadvantages of ORDBMS

- **Complexity**: The provision of the functionality that is expected of a good ORDBMS makes the ORDBMS an extremely complex piece of software. Database designers, developers, database administrators and end-users must understand this functionality to take full advantage of it. Failure to understand the system can lead to bad design decisions, which can have serious consequences for an organization.
- **Cost**: The cost of ORDBMS varies significantly, depending on the environment and functionality
  provided. There is also the recent annual maintenance cost.
- ORDBMS vendors are attempting to portray object models as extensions to the relational model with some additional complexity. This potentially misses the point of object orientation highlighting the large semantic gap between these two technologies.

Distributed Database System?

A distributed database (DDB) is a collection of multiple, *logically interrelated* databases
distributed over a *computer network.*

Distributed database management system (DDBMS)

A distributed database management system (D–DBMS) is the software that manages the
DDB and provides an access mechanism that makes this distribution transparent to the users.

Distributed database system (DDBS) = DDB + D–DBMS

Distributed Computing

□□□□A number of autonomous processing elements (not necessarily homogeneous) that are interconnected by a computer network and that cooperate in performing their assigned tasks.

What is distributed …

- Processing logic/ Processing elements
- Functions
- Data
- Control

Characteristics of DDBMS

- A collection of logically related shared data
- The data is splitted into a number of fragments
- Fragments may be replicated
- Fragments/replicas are allocated to sites
- The sites are linked by a communication network
- The data at each site is under the control of a DBMS
- The DBMS at each site can handle local applications , autonomously
- Each DBMS participates in at least one global application

Local and Global applications

- Users access the distributed database via applications. Applications are classified as those that do not require data from other sites(local applications) and that do require data from other sites(global applications)

Advantages DDBMS

- Reflects organizational structure
- Improved share ability and local autonomy
- Improved availability
- Improved reliability
- Improved performance
- Economics
- Modular growth

Disadvantages of DDBMS

- Complexity
- Cost
- Security
- Integrity control more difficult
- Lack of standards
- Lack of experience
- Database design more complex

What is not a DDBS?

- A timesharing computer system
- A loosely or tightly coupled multiprocessor system
- A database system which resides at one of the nodes of a network of computers - this is a centralized database on a network node

Applications of DDBS

- Manufacturing - especially multi-plant manufacturing
- Military command and control
- EFT
- Corporate MIS
- Airlines
- Hotel chains
- Any organization which has a decentralized organization structure

PARALLEL Vs. DISTRIBUTED TECHNOLOGY

Parallel Database Management Systems: DBMS developed using the following two types of multiprocessor system architectures are called parallel DBMS:

(i) **Shared memory(tightly coupled) architecture**: Multiple processors share secondary(disk) storage and also share primary memory.

(ii) Shared disk(loosely coupled) architecture: Multiple processors share secondary(disk) storage but each has their own primary memory.

Shared nothing architecture:

- In this architecture every processor has its own primary and secondary (disk) memory, no common

memory exists, and the processors communicate over a high speed interconnection network(bus or
switch).

- In shared nothing architecture, there is symmetry and homogeneity of nodes; this is not true for the

distributed database environment where heterogeneity of h/w and OS at each node is very common

- Shared nothing architecture is also considered as an environment for parallel databases

Deductive databases:

- In a deductive database system , we typically specify rules through a declarative language.
- A declarative language is a language in which we specify what to achieve rather than how

to achieve it.

- An inference engine (or deductive mechanism) within the system can deduce new facts from the

database by interpreting these rules.

- The model used for deductive databases is closely related to the relational data model, and

particularly to the domain relational calculus formalism.

- Deductive database is also related to the field of Logic Programming and Prolog language.
- The deductive database work based on logic has used Prolog as a starting point.
- A variation of Prolog called Datalog is used to define rules declaratively in conjunction with an

existing set of relations, which are themselves treated as literals in the language.

- A deductive database uses two main types of specifications: facts and rules.
- Facts are specified in a manner similar to the way relations are specified, except that it is not necessary to include the attribute names.
- In a deductive database, the meaning of an attribute value in a tuple is determined solely by its

position within the tuple.

- Rules are somewhat similar to relational views. They specify virtual relations that are not actually stored but that can be formed from the facts by applying inference mechanisms based on the rules specifications

Mobile databases:

- Mobile database is a database that is portable and physically separate from a centralized database server but is capable of communicating with that server from remote sites allowing the sharing of corporate data.
- With mobile databases, users have access to corporate data on their laptop, PDA, or other internet access device that is required for applications at remote sites.

The components of a mobile database environment: include:

- Corporate database server and DBMS that manages and stores the corporate data and provides
  corporate applications.
- Remote database and DBMS that manages and stores the mobile data and provides mobile applications .
- Mobile database platform that include Laptop, PDA, or other internet access devices.
- Two-way communication links between the corporate and mobile DBMS.

The additional functionality required for mobile database include the ability to:

- Communicate with the centralized database server through modes such as wireless or internet access.
- Replicate data on the centralized database server and mobile device.
- Synchronize data on the centralized database server and mobile device.
- Capture data from various sources such as internet.
- Manage data on the mobile device.
- Analyze data on the mobile device.
- Create customized mobile applications.

Multimedia databases:

- Multimedia databases provide features that allow users to store and query different types of multimedia information, which includes images(such as photos or drawings), video clips(such as movies, news reels, or home videos), audio clips(such as songs, phone messages, or speeches), and documents(such as books or articles).
- **Characteristics of multimedia sources:**
  - Image: An image is typically stored either in row form as asset of pixel or cell values, or in compressed form to save space.
  - Video: A video source is typically represented as a sequence of frames , where each frame is a still image.
  - Text/video source: A text/ video source is basically the full text of some article, book, or magazine.
  - Audio sources: It include stored recorded messages, such as speeches , class presentations, or even surveillance recording of phone messages.
- **Content based retrieval queries:** are those in which multimedia source is being retrieved based on its containing certain objects or activities.
- Multimedia database must ensure some model to organize and index the multimedia sources based on their contents.
- Identifying contents of multimedia sources is a difficult and time consuming task.
- **Two main approaches for identifying the contents of multimedia**
  - **Automatic analysis**
  - **Manual identification**
- **Automatic analysis:** Automatic analysis of the multimedia sources to identify certain mathematical characteristics of their contents. It uses different techniques depending on the type of multimedia source(image, text, video, or audio).

- **Manual identification:** Manual identification of the objects and activities of interest in each multimedia source and on using this information to index the sources. This approach can be applied to all the different multimedia sources, but it requires a manual preprocessing phase where a person has to scan each multimedia source to identify and catalog the objects and activities it contains so that they can be used to index these sources

Geographical Information System(GIS):

- GIS are used to collect, model, store, and analyze information describing physical properties of the geographical world.
- The scope of GIS broadly encompasses two types of data:
  - Spatial data
  - Non spatial data
- **Spatial data**: Spatial data originates from maps, digital images, administrative and political boundaries, roads, transportation networks; physical data such as rivers, soil characteristics, climate regions, land elevations
- **Non spatial data**: Non spatial data includes socio-economic data(like census counts), economic data, and sales or marketing information.
- **GIS applications**:
  - Cat graphic applications
  - Digital terrain modeling applications
  - Geographic objects applications
- **Cat graphic applications:**
  - Irrigation
  - Crop yield analysis
  - Land evaluation
  - Planning and facilities management
  - Landscape studies
  - Traffic pattern analysis
- **Digital terrain modeling applications**
  - Earth science studies
  - Civil engg. And military evaluation
  - Soil surveys
  - Air and water pollution studies
  - Flood control
  - Water resource management
- **Geographic objects applications**
  - Car navigation system
  - Geographic market analysis
  - Utility distribution and consumption
  - Consumer product and services economic analysis
- **Data management requirements of GIS:**
  - Data modeling and representation
  - Data analysis
  - Data integration
  - Data capture
- **Specific GIS data operations**
  - Interpolation
  - Interpretation
  - Proximity analysis

    - Raster image processing
    - Analysis of networks
    - Extensibility
    - Data quality control
    - Visualization
- **GIS software**: ARC-INFO