SCHOOL OF ENGINEERING

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

DATABASE MANAGEMENT SYSTEMS

2022-23

Contents

Chapter 1

Introduction to DBMS

1.1 Overview

Data are simply facts or figures, bits of information. When data are processed, interpreted, organized, structured or presented so as to make them meaningful or useful, they are called information. Information provides context for data.

Data usually refers to raw data, or unprocessed data. It is the basic form of data, data that hasn't been analysed or processed in any manner. Once the data is analysed, it is considered as information. Information is "knowledge communicated or received concerning a particular fact or circumstance." Information is a sequence of symbols that can be interpreted as a message. It provides knowledge or insight about a certain matter.

Some differences between data and information:

- Data is used as input for the computer system. Information is the output of data.
- Data is unprocessed facts figures. Information is processed data.
- Data doesn't depend on Information. Information depends on data.
- Data is not specific. Information is specific.
- Data is a single unit. A group of data which carries news and meaning is called Information.
- Data doesn't carry a meaning. Information must carry a logical meaning.
- Data is the raw material. Information is the product.

1.1.1 Database and Database Management System

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

For example: The college Database organizes the data about the admin, staff, students and faculty etc. Using the database, you can easily retrieve, insert, and delete the information.

Database Management System
A DBMS is a collection of inter related data and a set of programs to manipulate those data. Database management system is software which is used to manage the database.

DBMS = Database + set of programs

A database management system (DBMS) is a computerized system that enables users to create and maintain a database. The DBMS is a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications. Defining a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called meta-data. Constructing the database is the process of storing the data on some storage medium that is controlled by the DBMS. Manipulating a database includes functions such 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 simultaneously.

An application program accesses the database by sending queries or requests for data to the DBMS. A query typically causes some data to be retrieved; a transaction may cause some data to be read and some data to be written into the database. Other important functions provided by the DBMS include protecting the database and maintaining it over a long period of time. Protection includes system protection against hardware or software malfunction (or crashes) and security protection against unauthorized or malicious access. A typical large database may have a life cycle of many years, so the DBMS must be able to maintain the database system by allowing the system to evolve as requirements change over time.

There are many different types of database management systems, ranging from small systems that run on personal computers to huge systems that run on mainframes.

DBMS applications

There are different fields where a database management system is utilized. Following are a few applications which utilize the information base administration framework –

- Railway Reservation System
- Library Management System
- Banking
- Education Sector
- Credit card exchanges
- Social Media Sites
- Broadcast communications
- Account
- Online Shopping
- Human Resource Management
- Manufacturing

Examples of DBMS

- IBM DB2
- Microsoft Access
- Mango DB
- Microsoft SQL Server
- MySQL
- Oracle RDBMS

1.1.2 Function of DBMS

1. Defining database schema: it must give facility for defining the database structure also specifies access rights to authorized users
2. Manipulation of the database: The dbms must have functions like insertion of record into database updation of data, deletion of data, retrieval of data
3. Sharing of database: The DBMS must share data items for multiple users by main- taining consistency of data
4. Protection of database: It must protect the database against unauthorized

users 5.Database recovery: If for any reason the system fails DBMS must facilitate

database recovery

1.1.3 File oriented approach

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

Disadvantages of file oriented approach

1.1.3.1 Data redundancy and inconsistency:

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

1.1.3.2 Difficulty in accessing data:

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

1.1.3.3 Data isolation:

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

1.1.3.4 Integrity Problems:

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

1.1.3.5 Atomicity:

It is difficult to ensure atomicity in a file processing system when transaction failure occurs due to power failure, networking problems etc. (Atomicity: either all operations of the transaction are reflected properly in the database or non are)

1.1.3.6 Concurrent access:

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

1.1.3.7 Security problems:

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

1.2 File System vs DBMS

Basis	File System	DBMS
Structure	File system is a software that manages and organizes the files in a storage medium within a computer.	DBMS is a software for managing the database.
Data Redundancy	Redundant data can be present in a file system.	In DBMS there is no redundant data.
Backup and Recovery	It doesn't provide backup and recovery of data if it is lost.	It provides backup and recovery of data even if it is lost.
Query processing	There is no efficient query processing in file system.	Efficient query processing is there in DBMS.
Consistency	There is less data consistency in file system.	There is more data consistency because of the process of normalization
Complexity	It is less complex as compared to DBMS.	It has more complexity in handling as compared to file system.
Security Constraints	File systems provide less security in comparison to DBMS.	DBMS has more security mechanisms as compared to file system.
Cost	It is less expensive than DBMS.	It has a comparatively higher cost than a file system.

Table 1.1: Difference between File System and DBMS.

1.3 Advantages of DBMS

Data Independence

Data independence refers to characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

Reducing Data Redundancy

The file based data management systems contained multiple files that were stored in many different locations in a system or even across multiple systems. Because of this, there were sometimes multiple copies of the same file which lead to data redundancy. This is prevented in a database as there is a single database and any change in it is reflected immediately. Because of this, there is no chance of encountering duplicate data.

Sharing of Data

In a database, the users of the database can share the data among themselves. There are various levels of authorisation to access the data, and consequently the data can only be shared based on the correct authorisation protocols being followed. Many remote users can also access the database simultaneously and share the data between themselves.

Data Security

Data Security is vital concept in a database. Only authorised users should be allowed to access the database and their identity should be authenticated using a username and password. Unauthorised users should not be allowed to access the database under any circumstances as it violates the integrity constraints.

Privacy

The privacy rule in a database means only the authorized users can access a database according to its privacy constraints. There are levels of database access and a user can only view the data he is allowed to. For example - In social networking sites, access constraints are different for different accounts a user may want to access.

Backup and Recovery

Database Management System automatically takes care of backup and recovery. The users don't need to backup data periodically because this is taken care of by the DBMS. Moreover, it also restores the database after a crash or system failure to its previous condition.

Data Consistency

Data consistency is ensured in a database because there is no data redundancy. All data appears consistently across the database and the data is same for all the users viewing the database. Moreover, any changes made to the database are immediately reflected to all the users and there is no data inconsistency.

Disadvantage of DBMS:

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

1.4 Queries in DBMS

A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language. A DBMS provides a specialized language, called the query language, in which queries can be posed. A very attractive feature of the relational model is that it supports powerful query languages.

Different types of queries in DBMS are:

- Data definition language queries (DDL)
- Data manipulation language queries (DML)
- Data control language queries (DCL)
- Transaction language queries (TCL)

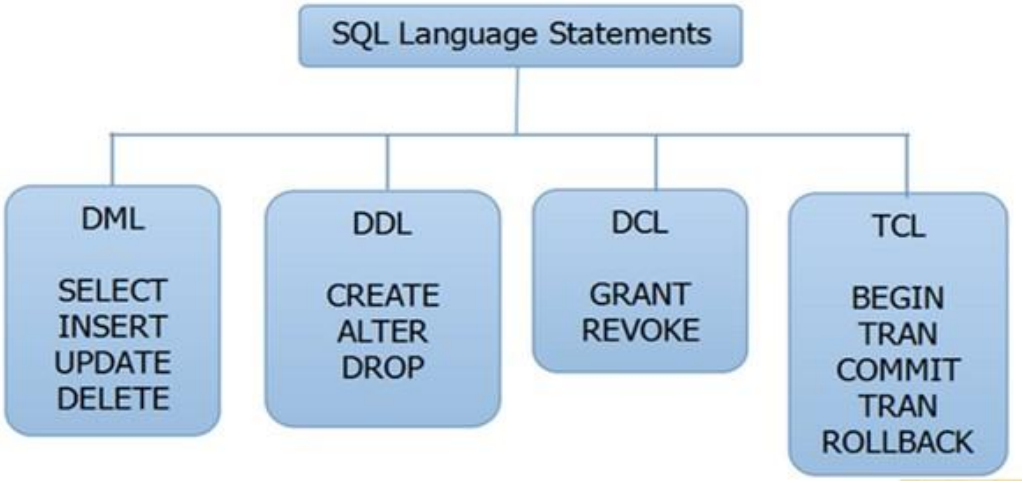


Figure 1.1: DBMS Queries

1.5 Transaction Management

A transaction is a set of logically related operations. For example, you are transferring money from your bank account to your friend's account, the set of operations would be:

- Read your account balance
- Deduct the amount from your balance
- Write the remaining balance to your account
- Read your friend's account balance
- Add the amount to his account balance
- Write the new updated balance to his account

These whole set of operations can be called as a transaction. The main problem that can happen during a transaction is that the transaction can fail before finishing the all the operations in the set. This can happen due to power failure, system crash etc. This

is a serious problem that can leave database in an inconsistent state. Assume that transaction fail after third operation (see the example above) then the amount would be deducted from your account but your friend will not receive it. To solve this problem, we have the following two operations:

1. **Commit:** If all the operations in a transaction are completed successfully then commit those changes to the database permanently.
2. **Rollback:** If any of the operation fails then rollback all the changes done by previous operations.

Even though these operations can help us avoiding several issues that may arise during transaction but they are not sufficient when two transactions are running concurrently. To handle those problems the database system maintains the ACID properties.

1. **Atomicity:** This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed.
2. **Consistency:** A transaction enforces consistency in the system state by ensuring that at the end of any transaction the system is in a valid state.
3. **Isolation:** For every pair of transactions, one transaction should start execution only when the other finished execution. I have already discussed the example of Isolation in the Consistency property above.
4. **Durability:** Once a transaction completes successfully, the changes it has made into the database should be permanent even if there is a system failure. The recovery-management component of database systems ensures the durability of transaction

Figure 1.2: Transaction state diagram

1. **Active:** In this state, the transaction is being executed. This is the initial state of every transaction.
2. **Partially Committed:** When a transaction executes its final operation, it is said to be in a partially committed state.
3. **Failed:** A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
4. **Aborted:** If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts

    - Re-start the transaction

- Kill the transaction

5. **Committed:** If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

1.6 DBMS Structure

- A database system is partitioned into modules that deal with each of the responsibilities of the overall system.
- The functional components of a database system can be broadly divided into the storage manager and the query processor components.
- The storage manager is important because databases typically require a large amount of storage space.
- The query processor is important because it helps the database system to simplify and facilitate access to data.
- It is the job of the database system to translate updates and queries written in a non-procedural language, at the logical level, into an efficient sequence of operations at the physical level.

Query Processor

The query processor components include

- DDL interpreter, which interprets DDL statements and records the definitions in the data dictionary.
- DML compiler, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
- A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs query optimization, that is, it picks the lowest cost evaluation plan from among the alternatives.
- Query evaluation engine, which executes low- level instructions generated by the DML compiler

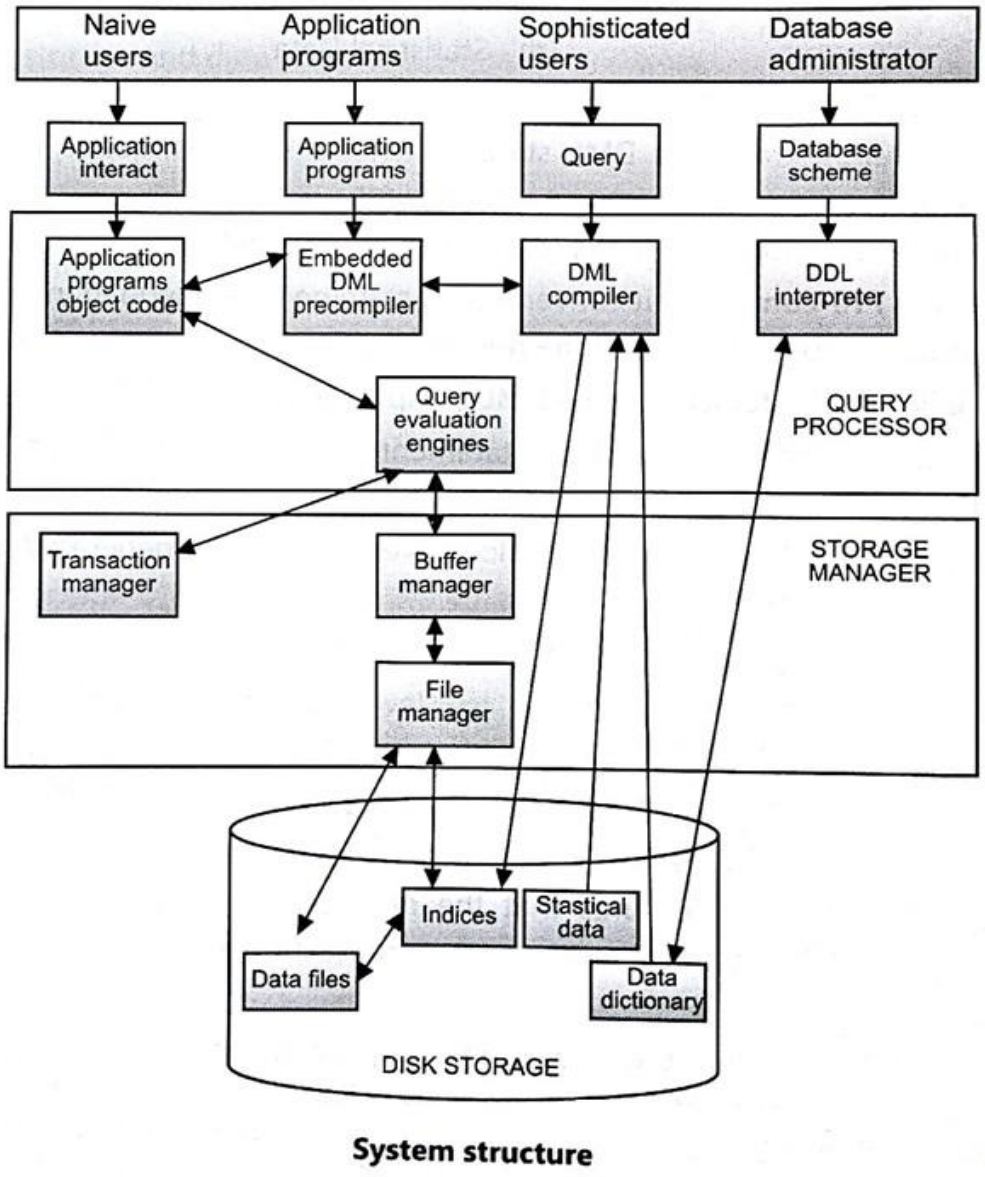


Figure 1.3: Structure of DBMS

1.6.3 Storage Manager

A storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The raw data are stored on the disk using the file system, which is usually provided by a conventional operating system. The storage manager translates the various DML statements into low- level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database. The storage manager components include:

Authorization and integrity manager: which tests for the satisfaction of integrity constraints and checks the authority of users to access data

Transaction manager: It's a component of DBMS which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.

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

Buffer manager, is responsible for fetching data from disk storage into main memory, and deciding what data to cache i main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

Transaction Manager, A transaction is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database-consistency constraints. That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates. Transaction manager ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.

1.6.4 Types of Database users

1.6.4.1 Database Administrator (DBA)

DBA is responsible for:

1.6.4.1.1 Deciding the instances for the database.

1.6.4.1.2 Defining the Schema

1.6.4.1.3 Liaising with Users

1.6.4.1.4 Define Security

1.6.4.1.5 Back-up and Recovery

1.6.4.1.6 Monitoring the performance

1.6.4.2 Database Designers

Database designers design the appropriate structure for the database, where we share data.

1.6.4.3 System Analyst

System analyst analyses the requirements of end users, especially naïve and parametric end users.

1.6.4.4 Application Programmers

Application programmers are computer professionals, who write application programs.

1.6.4.5 Naïve Users / Parametric Users

Naïve Users are Un-sophisticated users, which has no knowledge of the database. These users are like a layman, which has a little bit of knowledge of the database. Naive Users are just to work on developed applications and get the desired result. Example: Railway's ticket booking users are naive users. Or Clerical staff in any bank is a naïve user because they don't have any DBMS knowledge but they still use the database and perform their given task.

1.6.4.6 Sophisticated Users

Sophisticated users can be engineers, scientists, business analyst, who are familiar with the database. These users interact with the database but they do not write programs.

1.6.4.7 Casual Users / Temporary Users

These types of users communicate with the database for a little period of time.

1.7 Storage Data

A database system provides an ultimate view of the stored data. However, data in the form of bits, bytes get stored in different storage devices. In this section, we will take an overview of various types of storage devices that are used for accessing and storing data.

1.7.3 Types of Data Storage

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

- Primary Storage
- Secondary Storage
- Tertiary Storage

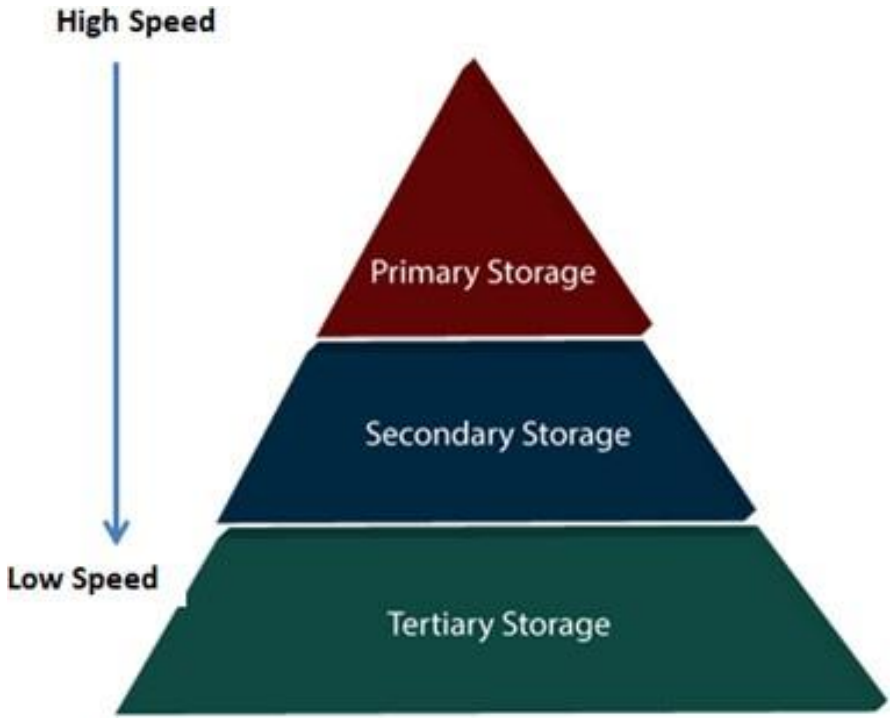


Figure 1.4: Types of data storage

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

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

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

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

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

1.7.3.2.1 **Flash Memory:** A flash memory stores data in USB (Universal Serial Bus) keys which are further plugged into the USB slots of a computer system. These USB keys help transfer data to a computer system, but it varies in size limits. Unlike the main memory, it is possible to get back the stored data which may be lost due to a power cut or other reasons. This type of memory storage is most commonly used in the server systems for caching the frequently used data. This leads the systems towards high performance and is capable of storing large amounts of databases than the main memory.

1.7.3.2.2 **Magnetic Disk Storage:** This type of storage media is also known as online storage media. A magnetic disk is used for storing the data for a long time. It is capable of storing an entire database. It is the responsibility of the computer system to make availability of the data from a disk to the main memory for further accessing. Also, if the system performs any operation over the data, the modified data should be written back to the disk. The tremendous capability of a magnetic disk is that it does not affect the data due to a system crash or failure, but a disk failure can easily ruin as well as destroy the stored data.

1.7.3.3 Tertiary Storage

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

There are following tertiary storage devices available:

1.7.3.3.1 **Optical Storage:** An optical storage can store megabytes or gigabytes of data. A Compact Disk (CD) can store 700 megabytes of data with a playtime of around 80 minutes. On the other hand, a Digital Video Disk or a DVD can store 4.7 or 8.5 gigabytes of data on each side of the disk.

1.7.3.3.2 **Tape Storage:** It is the cheapest storage medium than disks. Generally, tapes are used for archiving or backing up the data. It provides slow access to data as it accesses data sequentially from the start. Thus, tape storage is also known as sequential-access storage. Disk storage is known as direct-access storage as we can directly access the data from any location on disk.

1.7.4 Storage Hierarchy

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

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

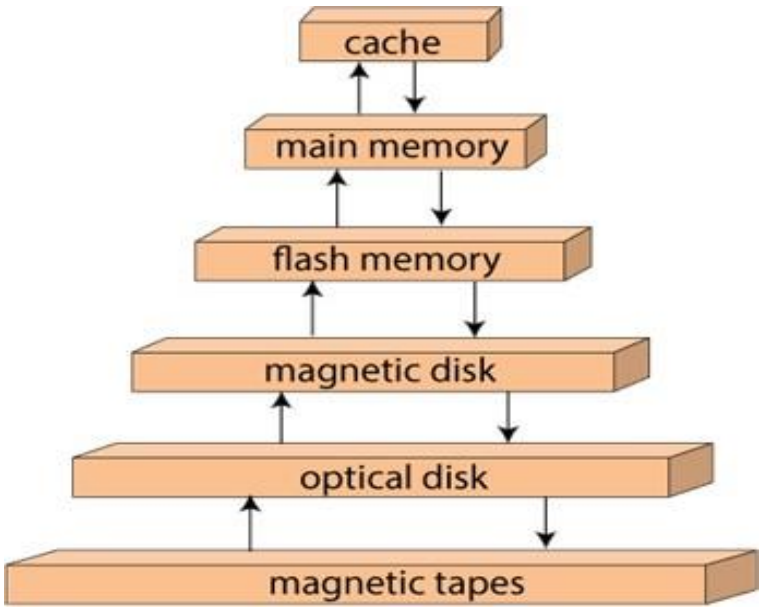


Figure 1.5: Storage device hierarchy

In the Figure.1.5, the higher levels are expensive but fast. On moving down, the cost per bit is decreasing, and the access time is increasing. Also, the storage media from the main memory to up represents the volatile nature, and below the main memory, all are non-volatile devices.

1.8 Data Models in DBMS

A DBMS allows a user to define the data to be stored in terms of a data model. A data model is a collection of high-level data description constructs that hide many low-level storage details. A data model is a collection of concepts that can be used to describe the structure of a database, we can categorize data models according to the types of concepts they use to describe the database structure.

- High-level or conceptual data models provide concepts that are close to the way many users perceive data.

- Low-level or physical data models provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks.

  Concepts provided by physical data models are generally meant for computer specialists, not for end users.

Between these two extremes is a class of Representational or implementation data models, which provide concepts that may be easily understood by end users.

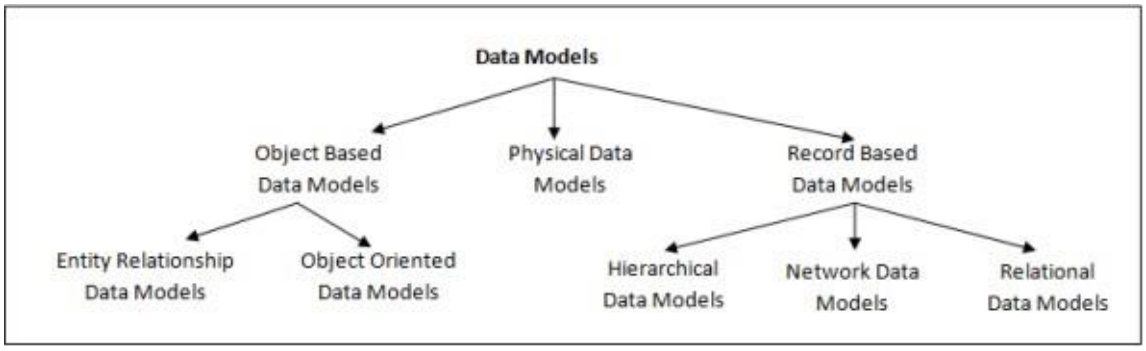


Figure 1.6: Data models classification

1.8.3 Entity - Relationship (E-R) Model

Conceptual data models or semantic data model is a more abstract, high-level data model that makes it easier for a user to come up with a good initial description of the data in an enterprise. A database design in terms of a semantic model serves as a useful starting point and is subsequently translated into a database design in terms of the data model the DBMS actually supports. A widely used semantic data model called the entity-relationship (ER) model allows us to pictorially denote entities and the relationships among them. It use concepts such as entities, attributes, and relationships.

An entity represents a real-world object or concept, such as an employee or a project from the world that is described in the database. An attribute represents some property of interest that further describes an entity, such as the employee's name or salary. A relationship among two or more entities represents an association among the entities, for example,

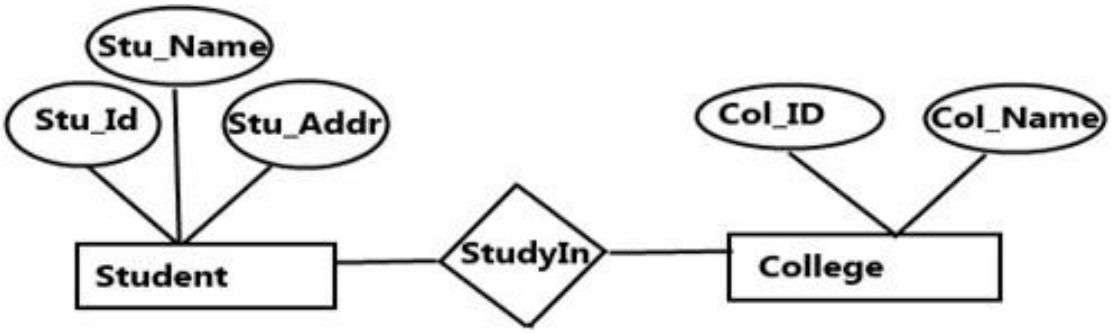


Figure 1.7: Sample E–R Diagram

Entity Relationship Model Advantages:

- Visual modelling yields conceptual simplicity
- Visual representation makes it an effective communication tool
- Is integrated with the dominant relational model

Disadvantages:

- Limited constraint representation
- Limited relationship representation
- No data manipulation language
- Loss of information content occurs when attributes are removed from entities to avoid crowded displays

1.8.4 Relational model

In relational database models, three key terms are used extensively: relations, attributes, and domains. A relation is a table with columns and rows. The named columns of the relation are called attributes, and the domain is the set of values the attributes are allowed to take.

Relational Model Advantages

- Structural independence is promoted using independent tables
- Tabular view improves conceptual simplicity
- Adhoc query capability is based on SQL
- Isolates the end user from physical-level details
- Improves implementation and management simplicity

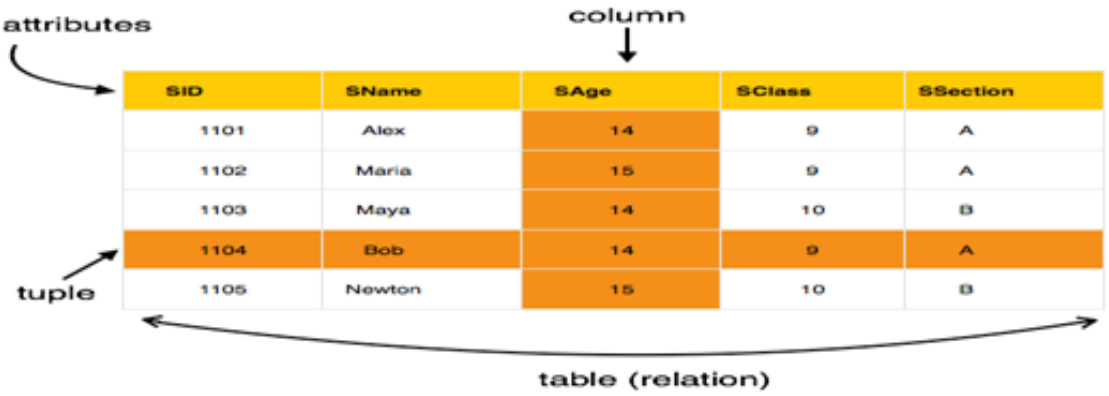


Figure 1.8: Sample Relational model

Disadvantages

- Requires substantial hardware and system software overhead
- Conceptual simplicity gives untrained people the tools to use a good system poorly
- May promote information problems

1.8.5 Hierarchical model

In a hierarchical model, data is organized into a tree-like structure, implying a single parent for each record. Hierarchical structures were widely used in the early mainframe database management systems, This structure allows one one-to-many relationship between two types of data. This structure is very efficient to describe many relationships in the real world. The main drawback of this model is that, it can have only one to many relationships between nodes.

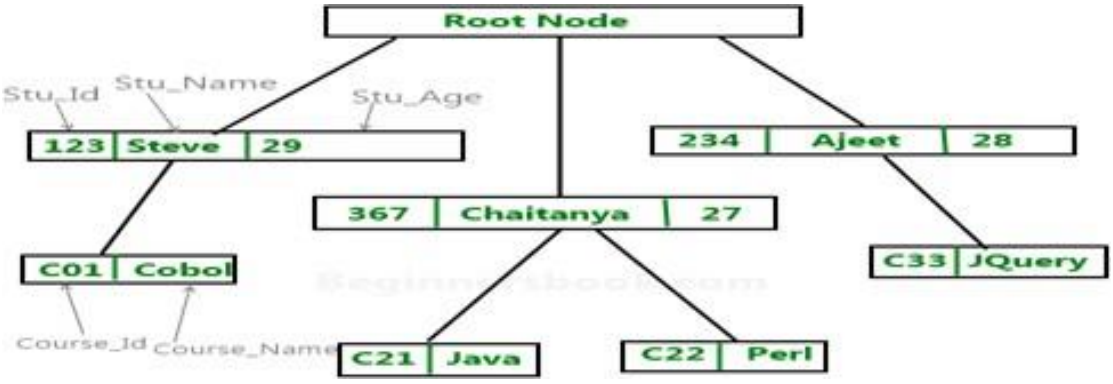


Figure 1.9: Sample Heirarchical model

Hierarchical Model Advantages

- Promotes data sharing
- Parent/child relationship promotes conceptual simplicity and data integrity
- Database security is provided and enforced by DBMS
- Efficient with 1:M relationships

Disadvantages

- Requires knowledge of physical data storage characteristics
- Navigational system requires knowledge of hierarchical path
- Changes in structure require changes in all application programs
- Implementation limitations
- No data definition
- Lack ofstandards

1.8.6 Network model

The network model expands upon the hierarchical structure, allowing many-to-many relationships in a tree-like structure that allows multiple parents. A record may be an owner in any number of sets, and a member in any number of sets. It was most popular before being replaced by the relational model. The network model is able to represent redundancy in data more efficiently than in the hierarchical model, and there can be more than one path from an ancestor node to a descendant.

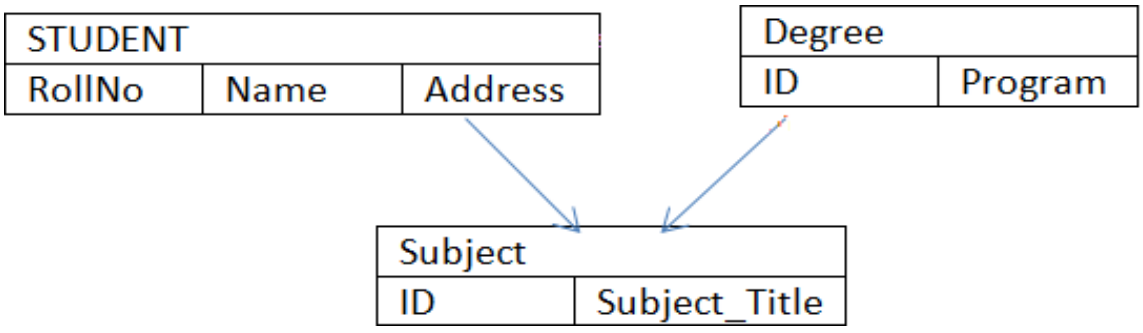


Figure 1.10: Sample Network model

Network Model Advantages

- Conceptual simplicity
- Handles more relationship types
- Data access is flexible
- Data owner/member relationship promotes data integrity
- Conformance to standards
- Includes data definition language (DDL) and data manipulation language (DML)

Disadvantages

- Navigational system yields complex implementation, application development, and management
- Structural changes require changes in all application programs

1.8.7 Object oriented data model

Object oriented data models are also frequently utilized as high-level conceptual models, particularly in the software engineering domain. Uses the E-R model as a basis but extended to include encapsulation, inheritance.

- Objects have both state and behaviour. State is defined by attributes. Behaviour is defined by methods (functions or procedures)
- Designer defines classes with attributes, methods, and relationships
- Class constructor method creates object instances
- Each object has a unique object ID
- Classes related by class hierarchies

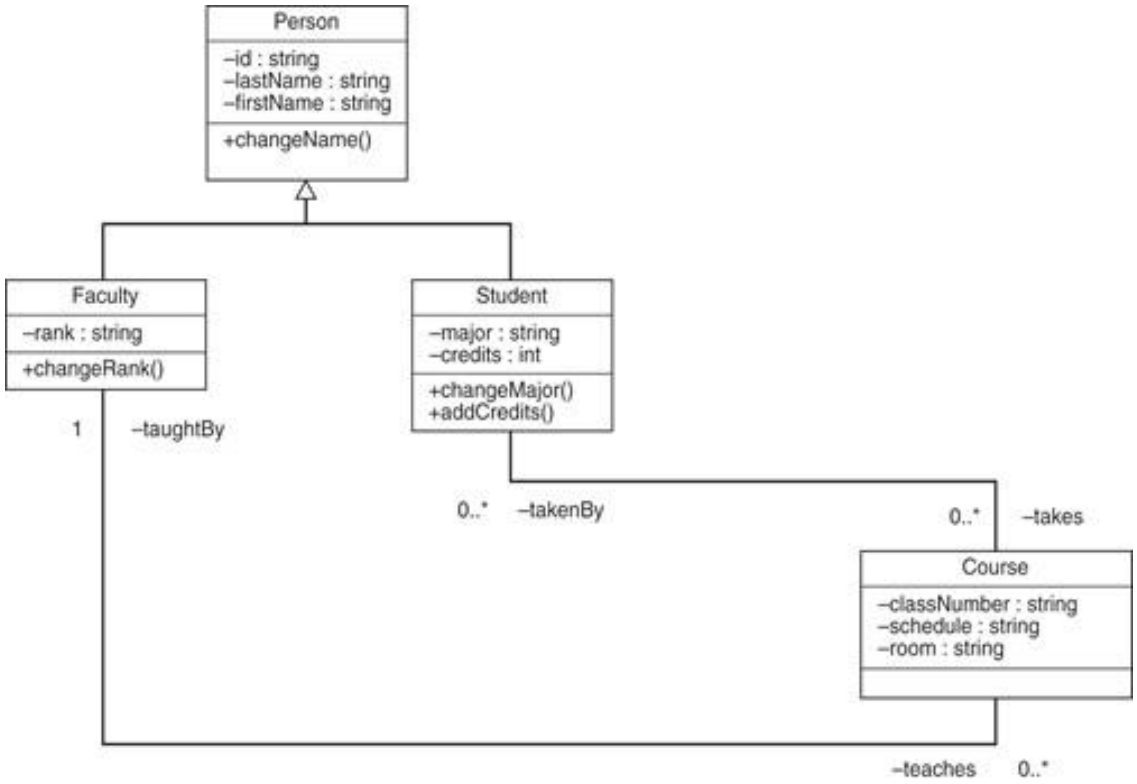


Figure 1.11: Sample Object Oriented model

Advantages

- Semantic content is added
- Visual representation includes semantic content
- Inheritance promotes data Integrity

Disadvantages

- Slow development of standards caused vendors to supply their own enhancements
- Compromised widely accepted standard
- Complex navigational system
- Learning curve is steep
- High system overhead slows transaction

1.8.8 Physical data model

Physical data models describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths. Physical data model represent the model where it describes how data are stored in computer memory, how they are scattered and ordered in the memory, and how they would be retrieved from memory. Basically physical data model represents the data at data layer or internal layer. It represents each table, their columns and specifications, constraints like primary key, foreign key etc. It basically represents how each tables are built and related to each other in DB.

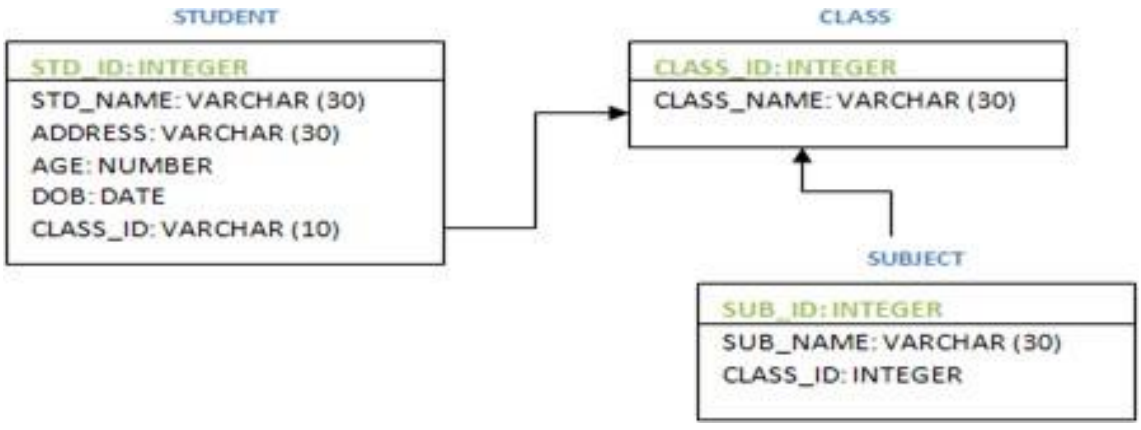


Figure 1.12: Sample Physical model

Figure1.12 shows how physical data model is designed. It is represented as UML diagram along with table and its columns. Primary key is represented at the top. The relationship between the tables is represented by interconnected arrows from table to table. Above STUDENT table is related to CLASS and SUBJECT is related to CLASS. The above diagram depicts CLASS as the parent table and it has 2 child tables – STUDENT and SUBJECT.

Importance of Data Models:

1.8.8.1 Are a communication tool. Data models can facilitate interaction among the de- signer, the applications programmer, and the end user.

1.8.8.2 Give an overall view of the database

1.8.8.3 Organize data for various users

1.8.8.4 Are an abstraction for the creation of good data base.

1.8.9 Evolution of Data Models

Evolution of Major Data Models

GENERATION	TIME	DATA MODEL	EXAMPLES	COMMENTS
First	1960s–1970s	File system	VMS/VSAM	Used mainly on IBM mainframe systems Managed records, not relationships
Second	1970s	Hierarchical and network	IMS, ADABAS, IDS-II	Early database systems Navigational access
Third	Mid-1970s	Relational	DB2 Oracle MS SQL Server MySQL	Conceptual simplicity Entity relationship (ER) modeling and support for relational data modeling
Fourth	Mid-1980s	Object-oriented Object/ relational (O/R)	Versant Objectivity/DB DB2 UDB Oracle 11g	Object/relational supports object data types Star Schema support for data warehousing Web databases become common
Fifth	Mid-1990s	XML Hybrid DBMS	dbXML Tamino DB2 UDB Oracle 11g MS SQL Server	Unstructured data support O/R model supports XML documents Hybrid DBMS adds object front end to relational databases Support large databases (terabyte size)
Emerging Models: NoSQL	Late 2000s to present	Key-value store Column store	SimpleDB (Amazon) BigTable (Google) Cassandra (Apache)	Distributed, highly scalable High performance, fault tolerant Very large storage (petabytes) Suited for sparse data Proprietary API

Figure 1.13: Evolution of data models

1.9 Data abstraction

It is a process of hiding unwanted or irrelevant details from the end user. It provides a different view and helps in achieving data independence which is used to enhance the security of data. Mainly there are three levels of abstraction for DBMS, which are as follows:

1. Physical or Internal Level

2. Logical or Conceptual Level

3. View or External Level

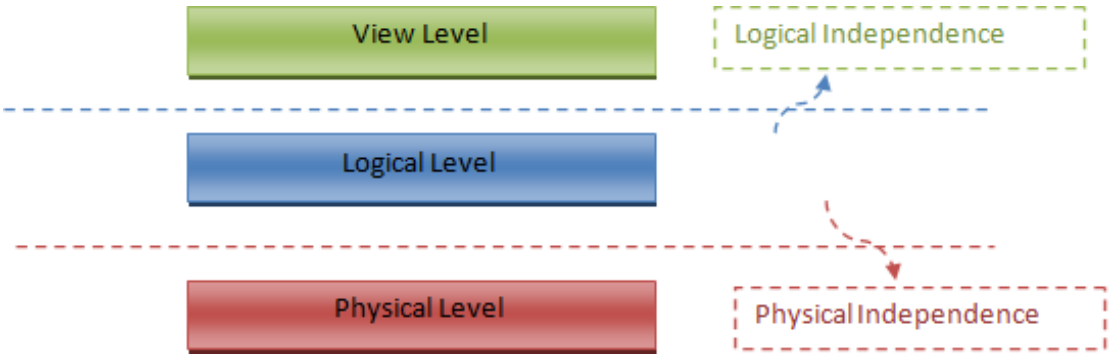


Figure 1.14: Data abstraction

Physical or Internal Level
It is the lowest level of abstraction for DBMS which defines how the data is actually stored, it defines data-structures to store data and access methods used by the database. Actually, it is decided by developers or database application programmers how to store the data in the database.

So, overall, the entire database is described in this level that is physical or internal level. It is a very complex level to understand. For example, customer's information is stored in tables and data is stored in the form of blocks of storage such as bytes, gigabytes etc.

Logical or Conceptual Level
Logical level is the intermediate level or next higher level. It describes what data is stored in the database and what relationship exists among those data. It tries to describe the entire or whole data because it describes what tables to be created and what are the links among those tables that are created.

It is less complex than the physical level. Logical level is used by developers or database administrators (DBA). So, overall, the logical level contains tables (fields and attributes) and relationships among table attributes.

View or External Level
It is the highest level. In view level, there are different levels of views and every view only defines a part of the entire data. It also simplifies interaction with the user and it provides many views or multiple views of the same database.

View level can be used by all users (all levels' users). This level is the least complex and easy to understand.

For example, a user can interact with a system using GUI that is view level and can enter details at GUI or screen and the user does not know how data is stored and what data is stored, this detail is hidden from the user.

Internal level or Physical level
It is the lowest level of abstraction and External or View level of abstraction is the highest level of abstraction. Based on these levels of abstraction, we have two types of data independence.

Physical Data Independence Physical Data Independence means changing the physical level without affecting the logical level or conceptual level. Using this property, we can change the storage device of the database without affecting the logical

schema. The changes in the physical level may include changes using the following

- A new storage device like magnetic tape, hard disk, etc.
- A new data structure for storage.
- A different data access method or using an alternative files organization technique.
- Changing the location of the database.

Logical Data Independence

Logical view of data is the user view of the data. It presents data in the form that can be accessed by the end users. Logical Data Independence says that users should be able to manipulate the Logical View of data without any information of its physical storage. Software or the computer program is used to manipulate the logical view of the data.

Database administrator is the one who decides what information is to be kept in the database and how to use the logical level of abstraction. It provides the global view of Data. It also describes what data is to be stored in the database along with the relationship. The data independence provides the database in simple structure. It is based on application domain entities to provide the functional requirement. It provides abstraction of system functional requirements. Static structure for the logical view is defined in the class object diagrams. Users cannot manipulate the logical structure of the database.

The changes in the logical level may include:

- Change the data definition.
- Adding, deleting, or updating any new attribute, entity or relationship in the database.

Chapter 2

Data Models

2.1 E-R diagram

ER diagram or Entity Relationship diagram is a conceptual model that gives the graphical representation of the logical structure of the database. It shows all the constraints and relationships that exist among the different components.

Components of E-R diagram
An ER diagram is mainly composed of following three components:

1.Entity Sets

2. Attributes

3.Relationship Set

Consider the following Student table. This complete table is referred to as "Student Entity Set" and each row represents an "entity".

Roll no	Name	Age
1	Akshay	20
2	Rahul	19
3	Pooja	20
4	Aarti	19

Table 2.1: Student table.

Representation as ER Diagram
The above table may be represented as ER diagram as shown in Figure.2.1. Here, Roll no is a primary key that can identify each entity uniquely. Thus, by using student's roll number, a student can be identified uniquely.

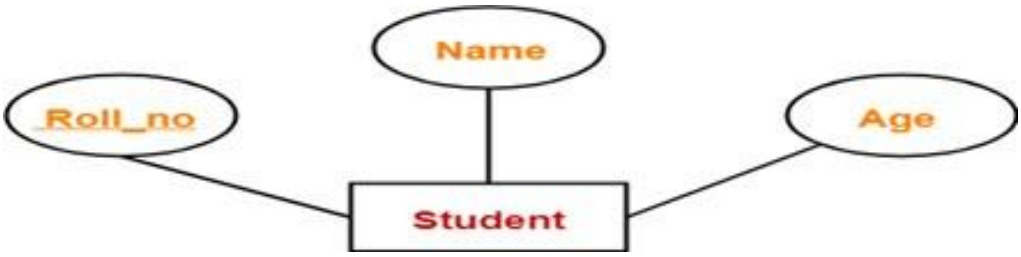


Figure 2.1: Student E-R diagram.

2.1.1 ER Diagram Symbols

An ER diagram is composed of several components and each component in ER diagram is represented using a specific symbol. ER diagram symbols are discussed below.

2.1.2 Entity sets

1. **Entity Sets**
   An entity set is a set of same type of entities. An entity refers to any object having -

   - Either a physical existence such as a particular person, office, house or car.
   - Or a conceptual/logical existence such as a school, a university, a company or a job.
   - Attributes are associated with an entity set.
   - Attributes describe the properties of entities in the entity set. Based on the values of certain attributes, an entity can be identified uniquely.

   An entity set may be of the following two types.

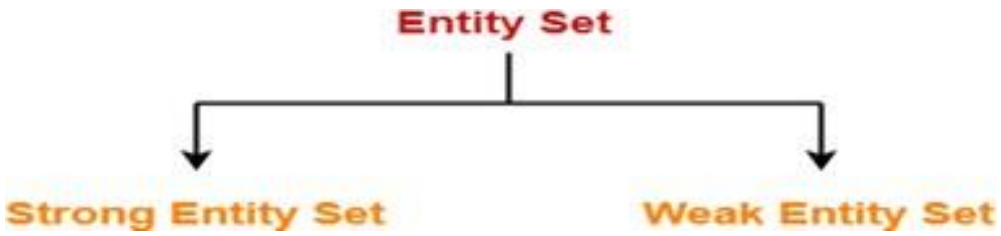


Figure 2.2: Entity sets.

A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities. In other words, a primary key exists for a strong entity set. Primary key of a strong entity set is represented by underlining it. It is represented using a single rectangle.

Figure 2.3: Strong and weak entity sets.

A weak entity set is an entity set that does not contain sufficient attributes to uniquely identify its entities. In other words, a primary key does not exist for a weak entity set. However, it contains a partial key called as a discriminator. Discriminator can identify a group of entities from the entity set. Discriminator is represented by underlining with a dashed line. It is represented using a double rectangle.

The combination of discriminator and primary key of the strong entity set makes it possible to uniquely identify all entities of the weak entity set. Thus, this combination serves as a primary key for the weak entity set as shown in Figure.2.4.

Figure 2.4: Primary key of a weak entity set.

A double rectangle is used for representing a weak entity set. A double diamond symbol is used for representing the relationship that exists between the strong and weak entity sets and this relationship is known as identifying relationship. A double line is used for representing the connection of the weak entity set with the relationship set. Total participation always exists in the identifying relationship.

Consider the following ER diagram shown in Figure.2.5. In this ER diagram

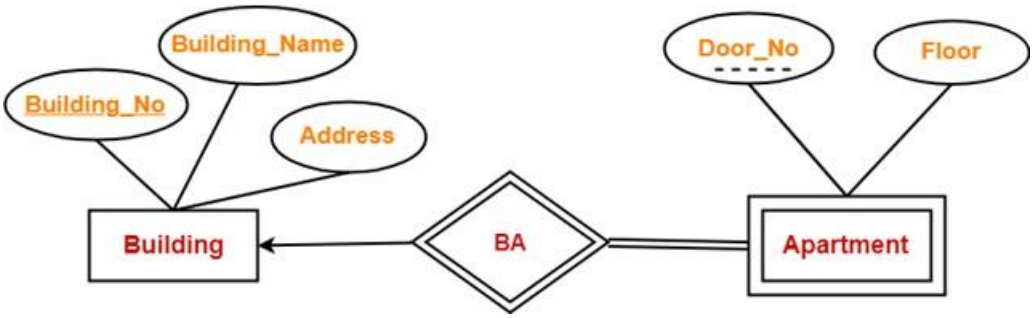


Figure 2.5: Example ER diagram.

- One strong entity set “Building” and one weak entity set “Apartment” are related to each other.
- Strong entity set “Building” has building number as its primary key.
- Door number is the discriminator of the weak entity set “Apartment”.

- This is because door number alone can not identify an apartment uniquely as there may be several other buildings having the same door number.
- Double line between Apartment and relationship set signifies total participation.
- It suggests that each apartment must be present in at least one building.
- Single line between Building and relationship set signifies partial participation.
- It suggests that there might exist some buildings which has no apartment.

2.1.3 Attributes

Attributes describe the properties of entities in an Entity Set. There exists a specific domain or set of values for each attribute from where the attribute can take its values.

The following are the types of attributes and the symbols are shown in Figure.2.6.

1.Simple attributes

2.Composite attributes

3.Single valued attributes

4.Multi valued attributes

5.Derived attributes

6.Key attributes

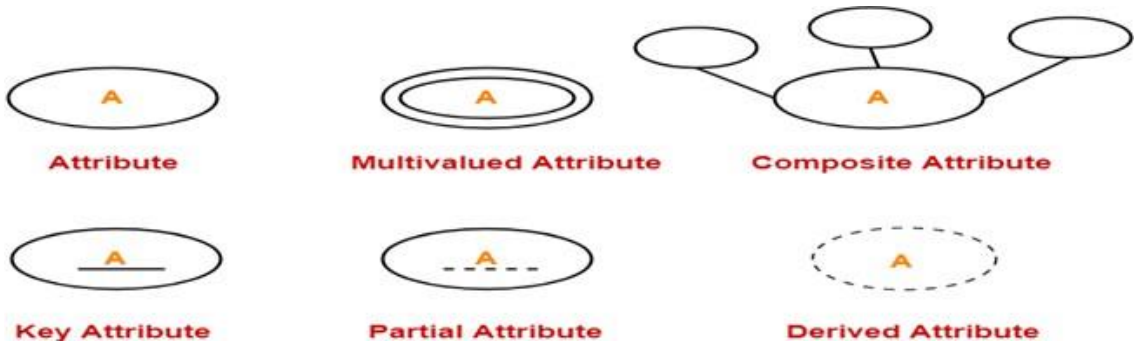


Figure 2.6: Attribute symbols used in E-R diagrams.

Simple Attributes
Simple attributes are those attributes which cannot be divided further. Here, all the attributes are simple attributes as they cannot be divided further. They are represented with an oval/ellipse.

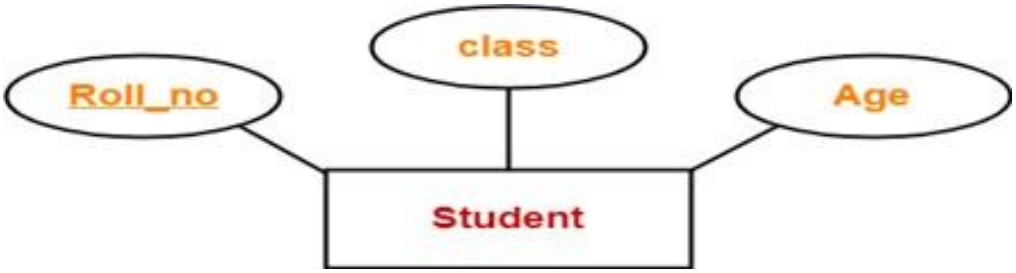


Figure 2.7: Simple attribute.

Composite Attributes

Composite attributes are those attributes which are composed of many other simple attributes.Here, the attributes "Name" and "Address" are composite attributes as they are composed of many other simple attributes.

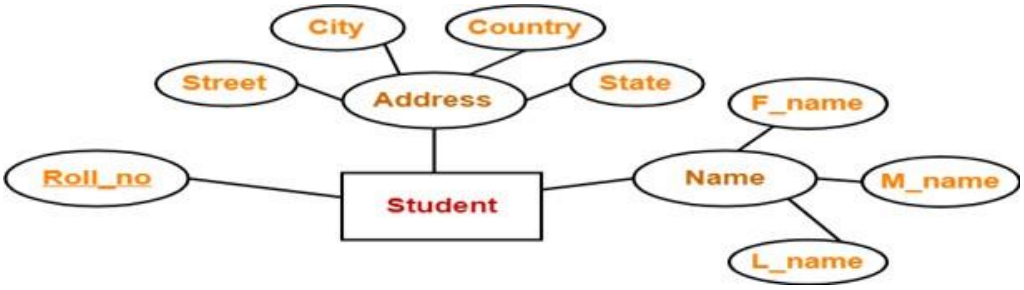


Figure 2.8: Composite attribute.

Single Valued Attributes

Single valued attributes are those attributes which can take only one value for a given entity from an entity set. Here, all the attributes are single valued attributes as they can take only one specific value for each entity. They are represented with an oval/ellipse.

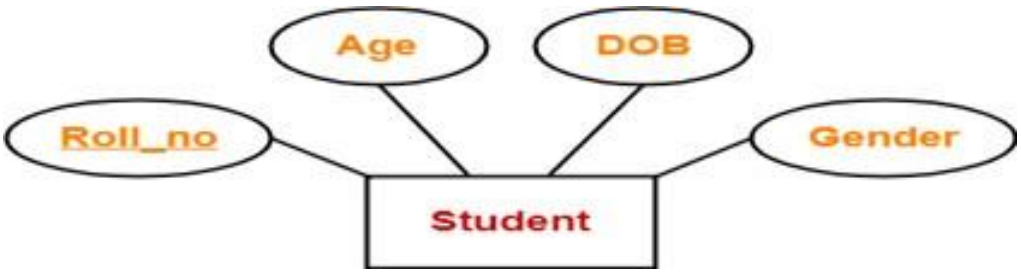


Figure 2.9: Single valued attribute.

Multi Valued Attributes

Multi valued attributes are those attributes which can take more than one value for a given entity from an entity set. They are represented with double oval/ellipse. Here, the attributes "Mob no" and "Email id" are multi valued attributes as they can take more than one values for a given entity.

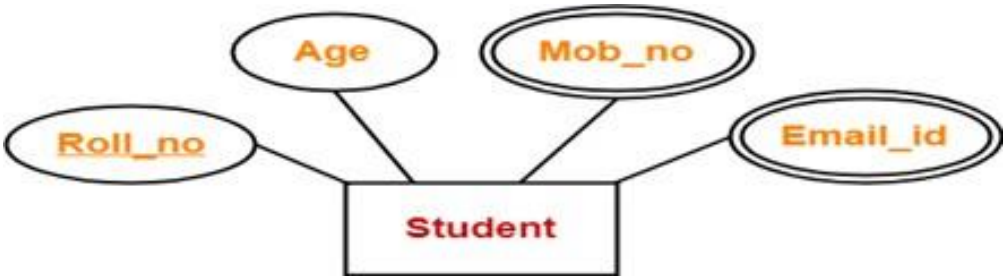


Figure 2.10: Multi valued attribute.

Derived Attributes
Derived attributes are those attributes which can be derived from other attribute(s). Here, the attribute "Age" is a derived attribute as it can be derived from the attribute "DOB". Derived attributes are represented with a dashed oval/ellipse.

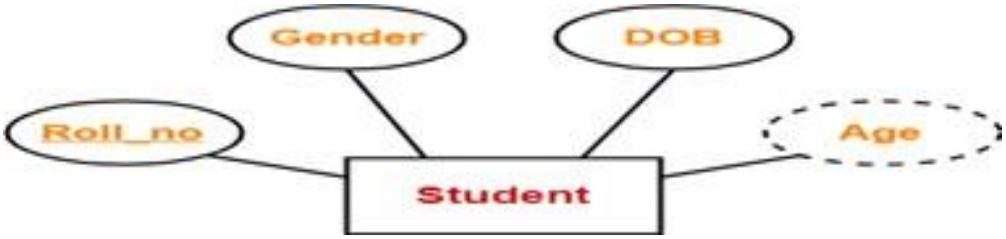


Figure 2.11: Derived attribute.

Key Attributes
Key attributes are those attributes which can identify an entity uniquely in an entity set. Here, the attribute "Roll no" is a key attribute as it can identify any student uniquely. It is represented with an underline in the oval/ellipse.

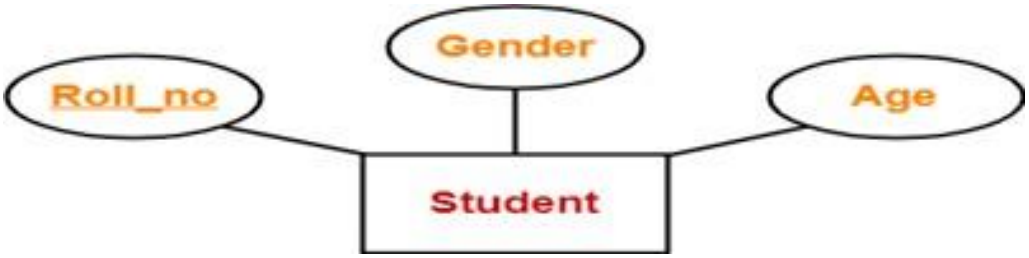


Figure 2.12: Key attribute.

2.1.4 Generalization, Specialization, and Aggregation

Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.

- In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.
- Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.

- In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass.

  For example, Faculty and Student entities can be generalized and create a higher level entity Person.

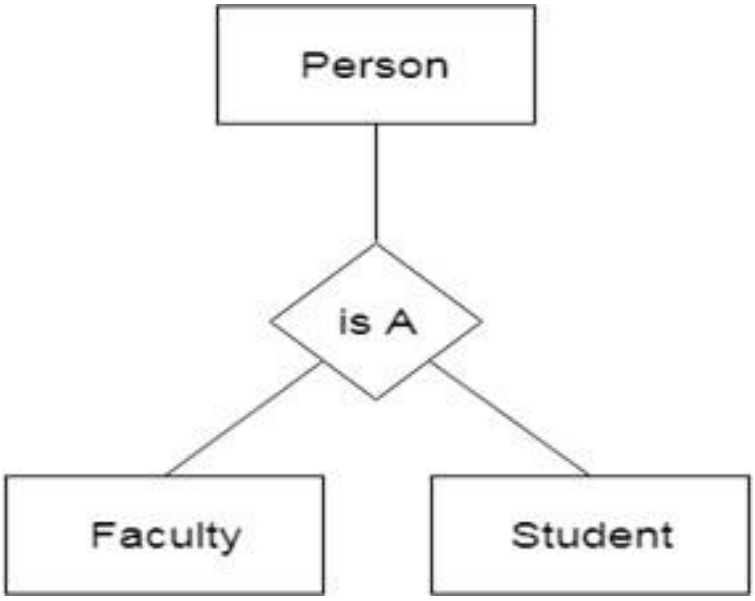


Figure 2.13: Generalization.

Specialization is a top-down approach, and it is opposite to Generalization.

- In specialization, one higher level entity can be broken down into two lower level entities.

- Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.

- Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

  For example: In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.

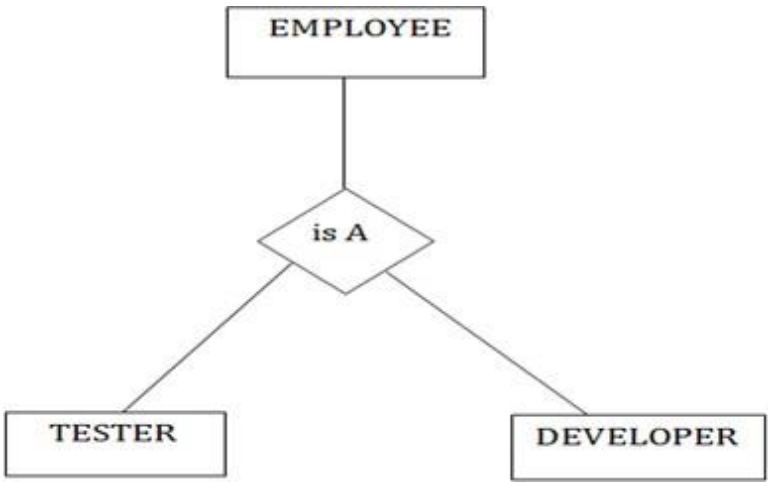


Figure 2.14: Specialization.

Specialization or generalization are represented in an E-R diagram as shown in Figure.2.15.

Figure 2.15: Specialization or generalization symbol.

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

For example: Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.

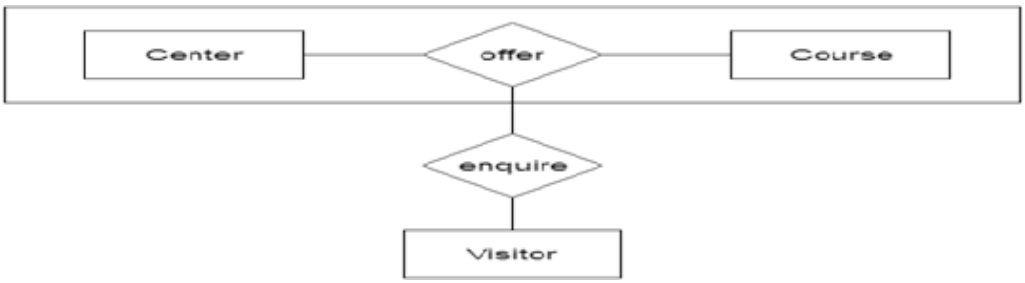


Figure 2.16: Aggregation.

2.1.5 Relationship Sets

Relationship defines an association among several entities. A relationship set is a set of same type of relationships. A relationship set may be of the following two types.

1.Strong relationship

- A strong relationship exists between two strong entity sets.
- It is represented using a diamond symbol.

2.Weak relationship

- A weak or identifying relationship exists between the strong and weak entity set.
- It is represented using a double diamond symbol.

Figure 2.17: Relationship set symbols in E-R diagram.

2.1.6 Participation Constraints

Participation constraint defines the least number of relationship instances in which an entity has to necessarily participate. There are two types of participation constraints and the symbols are as shown in Figure. **??**.

1.Partial participation
Partial participation is represented using a single line between the entity set and relationship set.

2. Total participation
Total participation is represented using a double line between the entity set and relationship set.

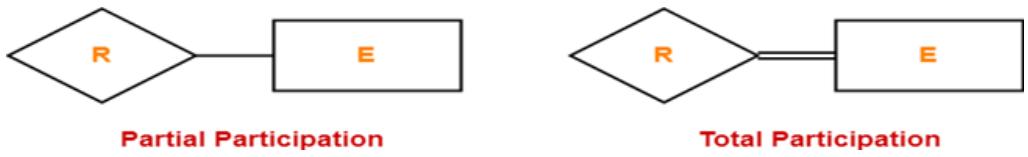


Figure 2.18: Participation symbols in E-R diagram.

2.1.7 Cardinality Constraints / Ratios

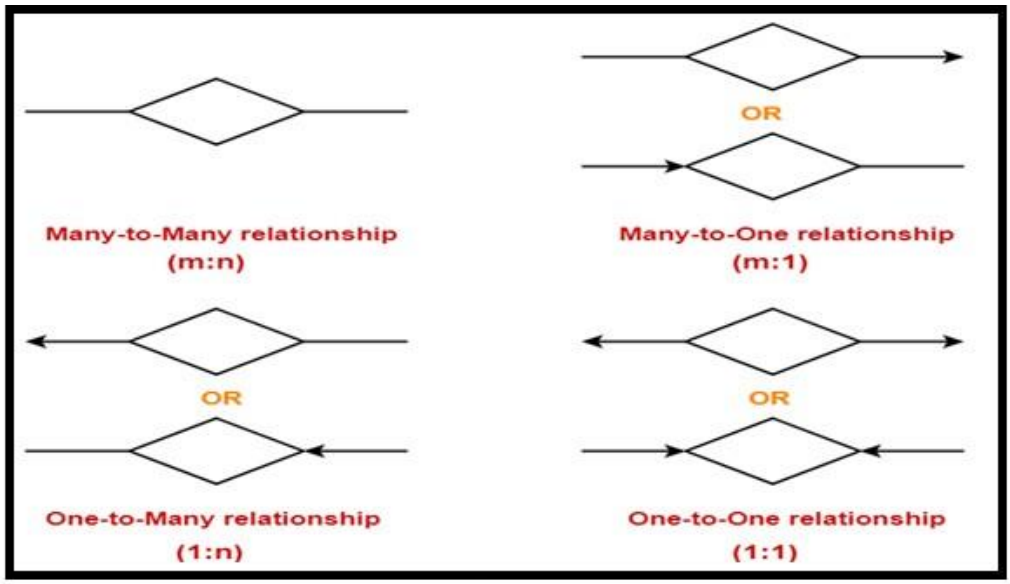
Cardinality constraint defines the maximum number of relationship instances in which an entity can participate. There are 4 types of cardinality ratios:

1.Many-to-many cardinality (m :n)

2.Many-to-one cardinality (m : 1)

3.One-to-many cardinality (1 : *n*)

4.One-to-one cardinality (1 : 1)



OR

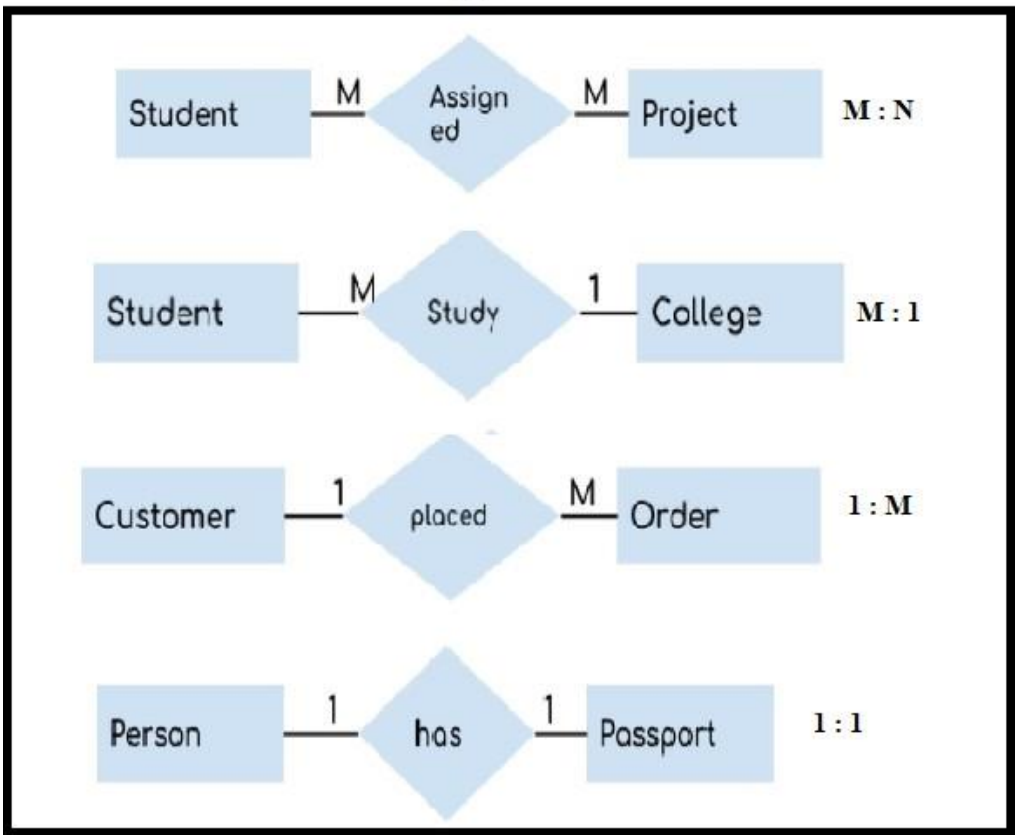


Figure 2.19: Relations in E-R diagram.

2.1.8 Relationship

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, areconnected to it by a line. A relationship is defined as an association among several

entities. Example: 'Enrolled in' is a relationship that exists between entities Student and Course.

Figure 2.20: Relations in E-R diagram.

Relationship Set

A relationship set is a set of relationships of same type. Set representation of ER diagram shown in Figure.2.20is represented as:

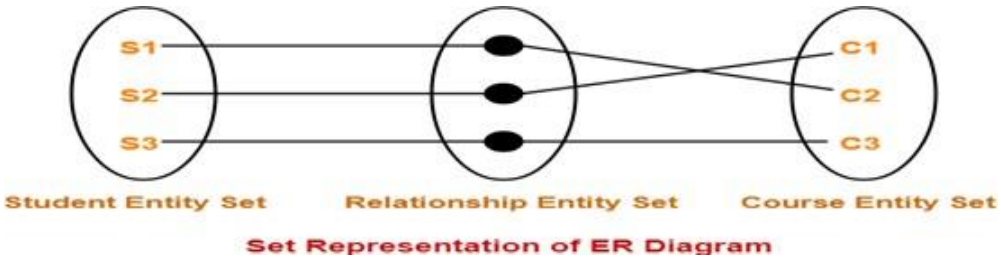


Figure 2.21: Relationship set in E-R diagram.

Degree of a Relationship Set

The number of entity sets that participate in a relationship set is termed as the degree of that relationship set. Thus,

Degree of a relationship set = Number of entity sets participating in a relationship set

Types of Relationship Sets

On the basis of degree of a relationship set, a relationship set can be classified into the following types

1. Unary relationship set
   Unary relationship set is a relationship set where only one entity set participates in a relationship set.

2. Binary relationship set
   Binary relationship set is a relationship set where two entity sets participate in a relationship set.

3. Ternary relationship set
   Ternary relationship set is a relationship set where three entity sets participate in a relationship set.

4. N-ary relationship set
   N-ary relationship set is a relationship set where 'n' entity sets participate in a relationship set.

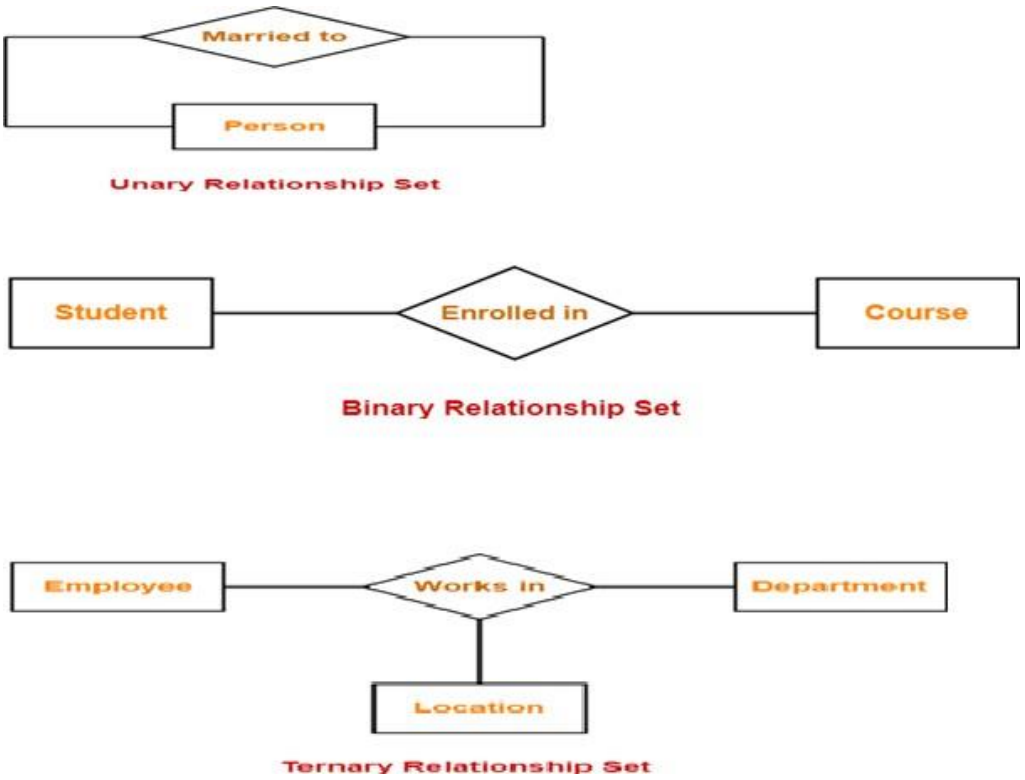


Figure 2.22: Relationship types in E-R diagram.

2.1.9 Participation Constraints

Participation constraints define the least number of relationship instances in which an entity must compulsorily participate. There are two types of participation constraints.

1. **Total Participation**
   It specifies that each entity in the entity set must compulsorily participate in at least one relationship instance in that relationship set. That is why, it is also called as mandatory participation. Total participation is represented using a double line between the entity set and relationship set.

Example:

Figure 2.23: Total participation in E-R diagram.

Here, double line between the entity set "Student" and relationship set "Enrolled in" signifies total participation. It specifies that each student must be enrolled in at least one course.

2. Partial Participation

It specifies that each entity in the entity set may or may not participate in the relationship instance in that relationship set. That is why, it is also called as optional participation. Partial participation is represented using a single line between the entity set and relationship set.

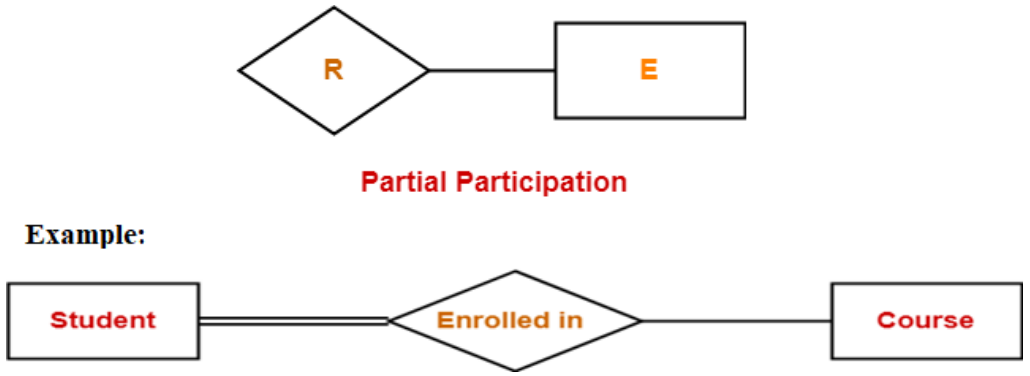


Figure 2.24: Partial participation in E-R diagram.

Here, Single line between the entity set "Course" and relationship set "Enrolled in" signifies partial participation. It specifies that there might exist some courses for which no enrollments are made.

Relationship between Cardinality and Participation Constraints

Minimum cardinality tells whether the participation is partial or total.

- If minimum cardinality = 0, then it signifies partial participation.
- If minimum cardinality = 1, then it signifies total participation.

Maximum cardinality tells the maximum number of entities that participates in a relationship set.

2.2 Relational Model

A database is a collection of 1 or more 'relations', where each relation is a table with rows and columns. The major advantages of the relational model over the older data models are:

1. It is simple and elegant.
2. Simple data representation.
3. The ease with which even complex queries can be expressed.

The main construct for representing data in the relational model is a 'relation'. A relation consists of following fields:

2.2.1 Relation Schema

The relation schema describes the column heads for the table. The schema specifies the relation's name, the name of each field (column, attribute) and the 'domain' of each field. A domain is referred to in a relation schema by the domain name and has a set of associated values.

Example: Student information in a university database to illustrate the parts of a relation schema. Students (Sid: string, name: string, login: string, age: integer, gross: real) this says that the field named 'sid' has a domain named 'string'. The set of values associated with domain 'string' is the set of all character strings.

2.2.2 Relation Instance

This is a table specifying the information. An instance of a relation is a set of 'tuples', also called 'records', in which each tuple has the same number of fields as the relation schemas. A relation instance can be thought of as a table in which each tuple is a row and all rows have the same number of fields. The relation instance is also called as 'relation'. Each relation is defined to be a set of unique tuples or rows.

Example: Fields (Attributes, Columns) Field names Tuples (Records, Rows) This example is an instance of the students relation, which consists 4 tuples and 5 fields. No two rows are identical.

sid	Name	login	age	Gross
1111	Dave	dave@cs	19	1.2
2222	Jones	Jones@cs	18	2.3
333	Smith	smith@ee	18	3.4
4444	Smith	smith@math	19	4

Figure 2.25: Relational database example.

Degree: The number of fields is called as 'degree'. This is also called as 'arity'. **Cardinality:** The cardinality of a relation instance is the number of tuples in it.

Example: In the above example, the degree of the relation is 5 and the cardinality is 4.

Relational database: It is a collection of relations with distinct relation names.

Relational database schema: It is the collection of schemas for the relations in the database. Instance: An instance of a relational database is a collection of relation instances, one per relation schema in the database schema. Each relation instance must satisfy the domain constraints in its schema.

2.2.3 Integrity constraints over relations

An integrity constraint (IC) is a condition that is specified on a database schema and restricts the data can be stored in an instance of the database. Various restrictions on data can be specified on a relational database schema in the form of 'constraints'. A DBMS enforces integrity constraints, in that it permits only legal instances to be stored in the database. Integrity constraints are specified and enforced at different times as below.

1. When the DBA or end user defines a database schema, he or she specifies the ICs that must hold on any instance of this database.
2. When a data base application is run, the DBMS checks for violations and disallows changes to the data that violate the specified ICs. Legal Instance: If the database instance satisfies all the integrity constraints specified on the database schema.

The constraints can be classified into 4 types as below.

1. **Domain Constraints**
   Domain constraints are the most elementary form of integrity constraints. They are tested easily by the system whenever a new data item is entered into the database. Domain constraints specify the set of possible values that may be associated with an attribute. Such constraints may also prohibit the use of null values for particular attributes. The data types associated with domains typically include standard numeric data types for integers A relation schema specifies the domain of each field or column in the relation instance. These domain constraints in the schema specify an important condition that each instance of the relation to satisfy: The values that appear in a column must be drawn from the domain associated with that column. Thus the domain of a field is essentially the type of that field.
2. **Key Constraints**
   A key constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for a tuple.

   Example: The 'students' relation and the constraint that no 2 students have the same student id (sid).

   Different keys:

   - Candidate Key or Key
   - Super Key
   - Primary Key
3. **Entity Integrity Constraints**
   This states that no primary key value can be null. The primary key value is used to identify individual tuples in a relation. Having null values for the primary key

implies that we cannot identify some tuples. NOTE: Key Constraints, Entity Integrity Constraints are specified on individual relations. PRIMARY KEYS comes under this.

4. **Referential Integrity Constraints**
   The Referential Integrity Constraint is specified between 2 relations and is used to maintain the consistency among tuples of the 2 relations. Informally, the referential integrity constraint states that 'a tuple in 1 relation that refers to another relation must refer to an existing tuple in that relation. We can diagrammatically display the referential integrity constraints by drawing a directed arc from each foreign key to the relation it references.

2.2.4 Keys in DBMS

Tables store a lot of data in them. Tables generally extend to thousands of records stored in them, unsorted and unorganized. Now to fetch any particular record from such dataset, you will have to apply some conditions, but what if there is duplicate data present and every time you try to fetch some data by applying certain condition, you get the wrong data. How many trials before you get the right data? To avoid all this, Keys are defined to easily identify any row of data in a table. Let's try to understand about all the keys using a simple example.

student_id	name	phone	Age
1	Akon	9876723452	17
2	Akon	9991165674	19
3	Bkon	7898756543	18
4	Ckon	8987867898	19
5	Dkon	9990080080	17

Figure 2.26: Keys in relational database example.

Super Key
Super Key is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a super set of Candidate key. In the table defined above super key would include studentid, (student id, name), phone etc.

The first one is pretty simple as student id is unique for every row of data, hence it can be used to identity each row uniquely. Next comes, (student id, name), now name of two

students can be same, but their student id can't be same hence this combination can also be a key. Similarly, phone number for every student will be unique, hence again, phone can also be a key. So they all are super keys.

Candidate Key
Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table. It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table. There can be more than one candidate key. In our example, student id and phone both are candidate keys for table Student.

- A candidate key can never be NULL or empty. And its value should be unique.
- There can be more than one candidate keys for a table.
- A candidate key can be a combination of more than one columns(attributes).

Primary Key
Primary key is a candidate key that is most appropriate to become the main key for any table. It is a key that can uniquely identify each record in a table.

For the table Student we can make the student id column as the primary key.

Composite Key
Key that consists of two or more attributes that uniquely identify any record in a table is called Composite key. But the attributes which together form the Composite key are not a key independently or individually.

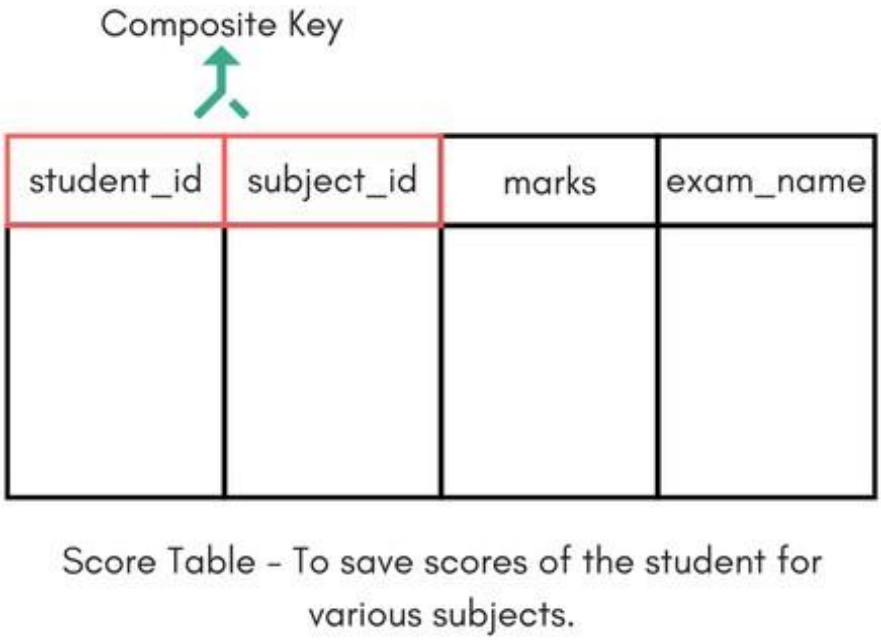


Figure 2.27: Composite keys in relational database example.

In the Figure.2.27 we have a Score table which stores the marks scored by a student in a particular subject. In this table student id and subject id together will form the primary key, hence it is a composite key.

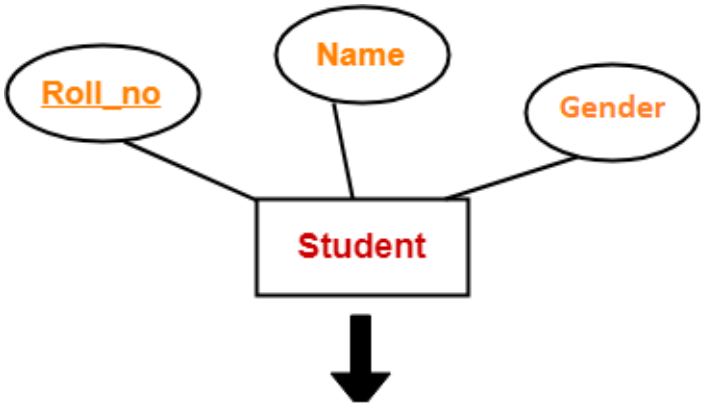
Secondary or Alternative key
The candidate key which are not selected as primary key are known as secondary keys or alternative keys.

2.3 E-R Diagram to Table Conversion

E-R diagram can be converted to a table by following the rules given below:

1. Strong Entity Set With Only Simple Attributes

- A strong entity set with only simple attributes will require only one table in relational model.
- Attributes of the table will be the attributes of the entity set.
- The primary key of the table will be the key attribute of the entity set.



Roll_no	Name	Gender

Schema : Student (Roll_no , Name , Gender)

Figure 2.28: E-R diagram to Table - Rule 1.

2. Strong Entity Set With Composite Attributes

- A strong entity set with any number of composite attributes will require only one table in relational model.
- While conversion, simple attributes of the composite attributes are taken into account and not the composite attribute itself.

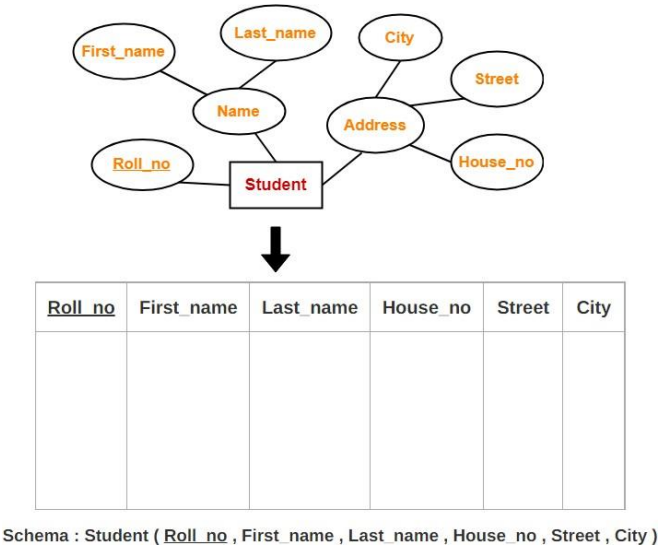


Figure 2.29: E-R diagram to Table - Rule 2.

3. Strong Entity Set With Multi Valued Attributes

- A strong entity set with any number of multi valued attributes will require two tables in relational model.
- One table will contain all the simple attributes with the primary key.
- Other table will contain the primary key and all the multi valued attributes.

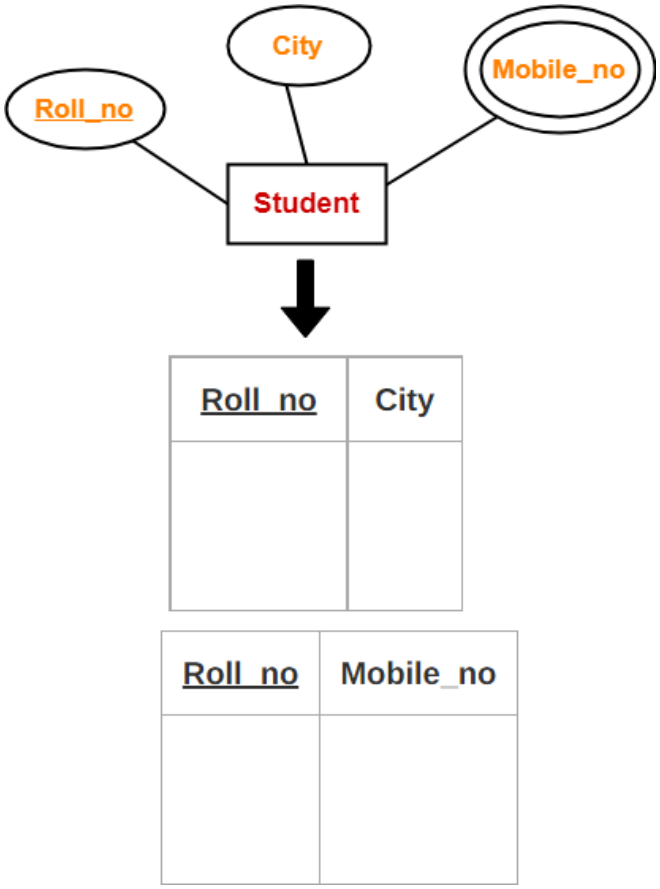
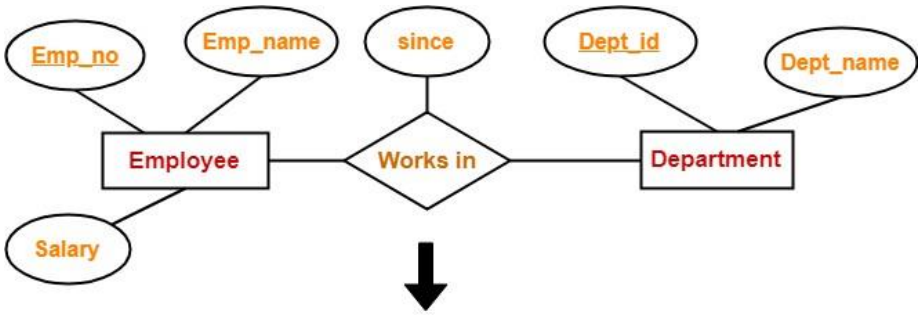


Figure 2.30: E-R diagram to Table - Rule 3.

4. Translating Relationship Set into a Table

- A relationship set will require one table in the relational model.
- Attributes of the table are:
  1. Primary key attributes of the participating entity sets
  2. Its own descriptive attributes if any.
  3. Set of non-descriptive attributes will be the primary key.



Emp_no	Dept_id	since

Schema : Works in (Emp_no , Dept_id , since)

Figure 2.31: E-R diagram to Table - Rule 4.

5. Binary Relationships With Cardinality Ratios

- **Case-01:** Binary relationship with cardinality ratio m:n

  – E1, R, E2 – 3 tables

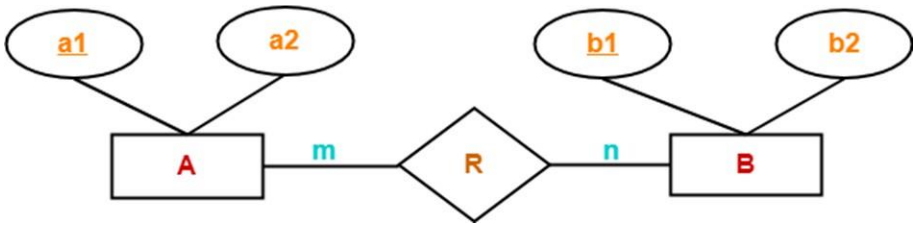


Figure 2.32: E-R diagram to Table - Rule 5(a).

This E-R diagram is translated into three tables

(a)A (a1 , a2)

(b) R (a1 , b1)

(c) B (b1 , b2)

- **Case-02:** Binary relationship with cardinality ratio 1:n

– E1, E2R - 2 tables

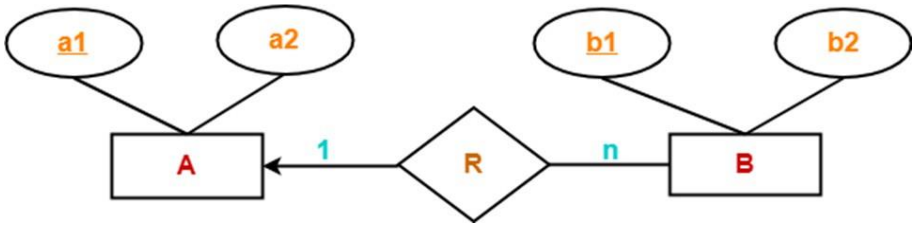


Figure 2.33: E-R diagram to Table - Rule 5(b).

This E-R diagram is translated into two tables

(a)A (a1 , a2)

(b)BR (a1 , b1 , b2)

- **Case-03:** Binary relationship with cardinality ratio m:1

  – E1R, E2 - 2 tables

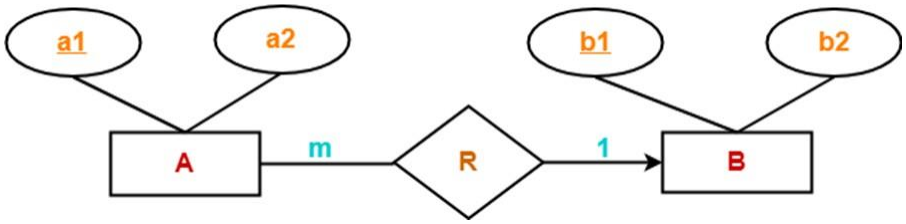


Figure 2.34: E-R diagram to Table - Rule 5(c).

This E-R diagram translates to two tables

(a)AR (a1 , a2 , b1)

(b)B (b1 , b2)

- **Case-04:** Binary relationship with cardinality ratio 1:1

  – E1R, E2 orE1, E2R - 2 tables

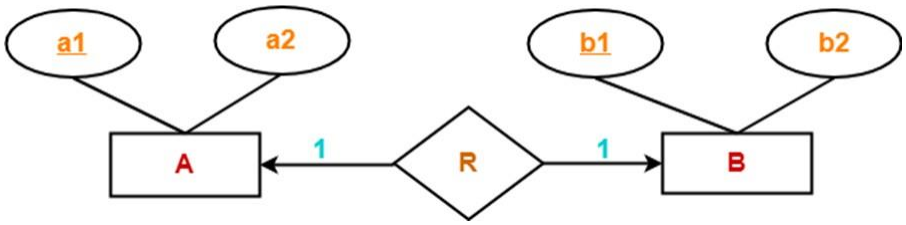


Figure 2.35: E-R diagram to Table - Rule 5(d).

This E-R diagram translates to two tables

(a)AR (a1 , a2 , b1)

(b)B (b1 , b2)

(OR)

(a) A (a1 , a2)

(b) BR (a1 , b1 , b2)

6. Binary Relationship With Both Cardinality Constraints and Participation Constraints

- Cardinality constraints will be implemented as discussed in Rule-05.
- Total participation constraint ⇒ foreign key acquires NOT NULL constraint i.e. now foreign key can not be null.
- **Case-01:** Binary Relationship With Cardinality Constraint and Total Participation Constraint From One Side
  E1, E2R – with a foreign key constraint

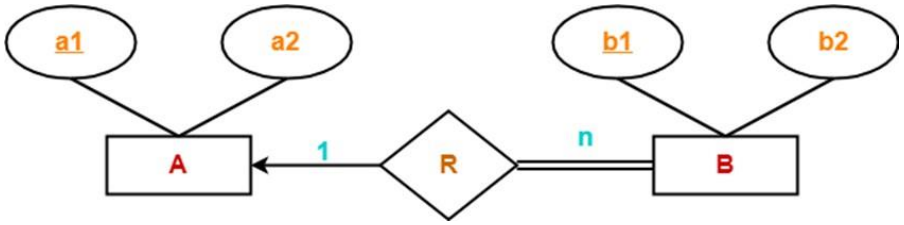


Figure 2.36: E-R diagram to Table - Rule 6(a).

This E-R diagram translates to Two tables.

(a)A (a1 , a2)

(b)BR (a1 , b1 , b2)

- **Case-02:** Binary Relationship With Cardinality Constraint and Total Participation Constraint From Both Sides
  E1R E2 – One table

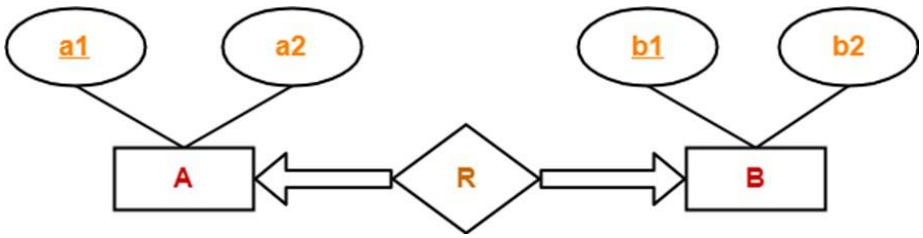


Figure 2.37: E-R diagram to Table - Rule 6(b).

This E-R diagram translates to One tables.

(a)ARB (a1 , a2 , b1 , b2)

7. **Binary Relationship With Weak Entity Set** Weak entity set always appears in association with identifying relationship with total participation constraint.

- E1, E2R – Two tables

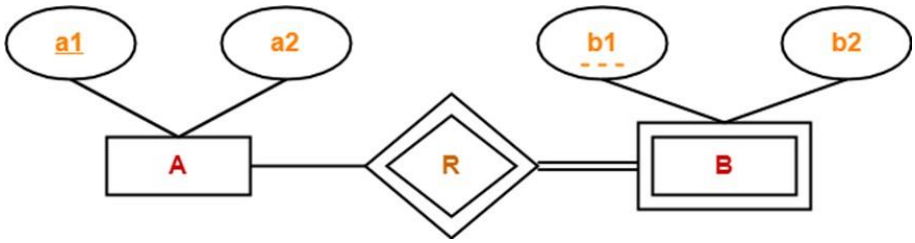


Figure 2.38: E-R diagram to Table - Rule 7.

This E-R diagram translates to Two tables.

(a)A (a1 , a2)

(b)BR (a1 , b1 , b2)

Example: E-R diagram to Table Conversion

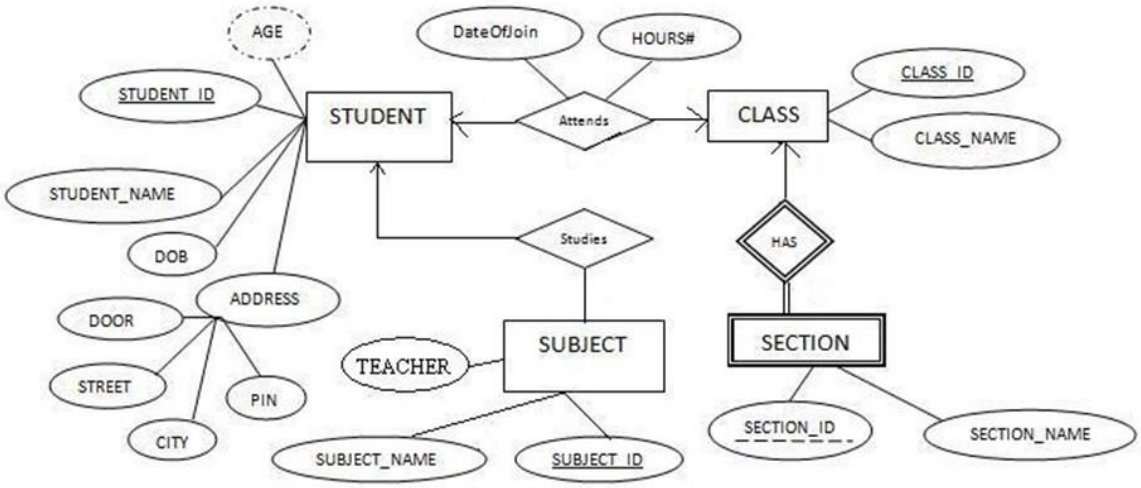


Figure 2.39: E-R diagram to Table - Example problem.

The final database contains the following tables:

1. STUDENT (StudentID, StudentName, DOB, Door, Street, City, Pin)
2. CLASS (ClassID, ClassName, StudentID, DateOfJoin, Hours)
    - StudentID is the foreign key refers STUDENT table
3. SUBJECT (SubjectID, SubjectName, Teacher, StudentID)
    - StudentID is the foreign key refers STUDENT table
4. SECTION (SectionID, ClassID, SectionName)
    - ClassID is the foreign key refers CLASS table

ER components	Given component	Result
Strong Entity Set *Rule: Strong entity set can be directly converted into table.*	(a) STUDENT (b) SUBJECT (c) CLASS	(a) STUDENT (Student_ID, Student_Name, DOB, Address) (b) SUBJECT (Subject_ID, Subject_Name, Teacher) (C) CLASS (Class_ID, Class_Name)
Derived attribute *Rule: No need to create a column in the table for derived attribute.*	Age in STUDENT table	No changes
Composite attribute *Rule: Replace the composite attribute with its component attributes.*	Address in STUDENT table	**STUDENT (Student_ID, Student_Name, DOB, Door, Street, City, Pin)**
1-1, 1-n, and n-1 Relationships *Rule: Include the primary key of one side entity set as the foreign key of other side entity set.*	Attends (1-1 from STUDENT to CLASS) Studies (1-n from STUDENT to SUBJECT)	CLASS (Class_ID, Class_Name, Student_ID) **SUBJECT (Subject_ID, Subject_Name, Teacher, Student_ID)**
Descriptive attribute *Rule: An attribute that is part of a relationship is descriptive. Include the descriptive attributes to 1 side as shown above.*	DateOfJoin, Hours# of Attends relationship.	**CLASS (Class_ID, Class_Name, Student_ID, DateOfJoin, Hours#)**
Weak entity set *Rule: Weak entity set is totally participated (existence dependent) on the strong entity set. Include the primary key of strong entity set into the weak entity set as foreign key.*	(d) SECTION	**SECTION (Section_ID, Section_Name, Class_ID)**

Figure 2.40: E-R diagram to Table - Example problem Solution.

2.3.1 Views (Virtual Table)

A view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition. Views are a logical virtual table created by "select query" but the result is not stored anywhere in the disk and every time we need to fire the query when we need data, so always we get updated or latest data from original tables.

1. A view is a predefined query on one or more tables.

2. Retrieving information from a view is done in the same manner as retrieving from a table.

3. With some views you can also perform DML operations (delete, insert, update) on the base tables.

4. Views don't store data, they only access rows in the base tables.

5. User tables, user sequences, and user indexes are all views.

6. View can hide the underlying base tables.

7. By writing complex queries as a view, we can hide complexity from an end user.

8. View only allows a user to access certain rows in the base tables.

Creating Views
A View can be created from a single table or multiple tables.

```
CREATE VIEW view _name
AS SELECT column1, column2,.. columnn
FROM table_name WHERE condition;
```

Listing 2.1: Creating views

Student Data (SD) Table

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

Student Marks (SM) Table

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Figure 2.41: Creating views.

```
CREATE VIEW Details  AS
 SELECT NAME , ADDRESS
FROM SD WHERE S_ ID < 5;
```

Listing 2.2: Creating view from a single table

To see the data in the View, we can query the view in the same manner as we query a table. The output is shown in Figure.2.42.

```
SELECT * FROM Details;
```

Listing 2.3: See the data in views

S_ID	NAMES
2	Ashish
4	Dhanraj
1	Harsh
3	Pratik
5	Ram

Figure 2.42: Output of one table view.

Creating View from multiple tables

```
CREATE VIEW Marks AS
SELECT SD.NAME, SD.ADDRESS, SM.MARKS FROM
 SD, SM WHERE SD.NAME = SM.NAME;
```

Listing 2.4: Creating view from a multiple tables

NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

Figure 2.43: Output of multi table view.

Updating Views

A view can be updated under certain conditions which are given below:

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.

We can use the CREATE OR REPLACE VIEW statement to add or remove fields from a view.

Create OR Replace View

```
view_name AS SELECT column1,coulmn2,..
FROM table_name
WHERE condition ;
```

Listing 2.5: Create or replace view

For example, if we want to update the view Marks and add the field AGE to this View from SM Table, we can do this as:

```
CREATE OR REPLACE VIEW  Marks

AS SELECT SD.NAME, SD.ADDRESS, SM.MARKS,SM.
AGE FROM SD,SM WHERE SD.NAME = SM.NAME;
```

Listing 2.6: Create or replace view

INSERTING a row

```
INSERT INTO View name(C1,C2...Cn)
VALUES(V1,V2    .Vn);
INSERT INTO Details(NAME, ADDRESS) VALUES("Suresh","Gurgaon");
```

Listing 2.7: Inserting a row

DELETING Rows

Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.

```
DELETE FROM view _ name WHERE condition;
DELETE FROM Details WHERE NAME =" Suresh";
```

Listing 2.8: Deleting a row

DELETING Views

We can delete or drop a View using the DROP statement.

```
DROP VIEW view_name;
```

Listing 2.9: Deleting a row

Materialized View

Materialized views are also the logical view of our data-driven by the select query but the result of the query will get stored in the table or disk.

2.4 Relational Algebra

Relational Algebra is procedural query language, which takes Relation as input and generate relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL. Each relational query describes a step-by-step procedure for computing the desired answer, based on the order in which operators are applied in the query.

2.4.1 Importance of Relational Algebra

First, it provides a formal foundation for relational model operations.

Second, and perhaps more important, it is used as a basis for implementing and optimizing queries in the query processing and optimization modules that are integral parts of relational database management systems (RDBMS).

Third, some of its concepts are incorporated into the SQL standard query language for RDBMS.

The fundamental operation included in relational algebra are Select (σ), Project (π), Union ($\cup$), Set Difference ($-$), Cartesian product ($\times$) and Rename (ρ)

2.4.2 Relational Calculus

Unlike Relational Algebra, Relational Calculus is a higher level Declarative language. In converse to the relational algebra, relational calculus defines what result is to be obtained. Like Relational Algebra, Relational Calculus does not specify the sequence of operations in which query will be evaluated.

The sequence of relational calculus operations is called relational calculus expression that also produces a new relation as a result. The Relational Calculus has two variations namely

1. Tuple Relational Calculus and
2. Domain Relational Calculus.

The Tuple Relational Calculus list the tuples to selected from a relation, based on a certain condition provided. It is formally denoted as:

$$\{t|P\,t\} \tag{2.1}$$

Where t is the set of tuples for which the condition P is true.

The next variation is Domain Relational Calculus, which in contrast to Tuple Relational Calculus list the attributes to be selected from a relation, based on certain condition. The formal definition of Domain Relational Calculus is as follows:

$$\{< X_1, X_2, X_3, \ldots X_n > | P X_1, X_2, X_3, \ldots X_n\} \tag{2.2}$$

Where $X_1, X_2, X_3, \ldots X_n$ are the attributes and P is the certain condition.

Key Differences Between Relational Algebra and Relational Calculus

- The basic difference between Relational Algebra and Relational Calculus is that Relational Algebra is a Procedural language whereas, the Relational Calculus is a Non-Procedural, instead it is a Declarative language.
- The Relational Algebra defines how to obtain the result whereas, the Relational Calculus define what information the result must contain.
- Relational Algebra specifies the sequence in which operations have to be performed in the query. On the other hands, Relational calculus does not specify the sequence of operations to performed in the query.
- The Relational Algebra query language is closely related to programming language whereas; the Relational Calculus is closely related to the Natural Language.

Relational Algebra and Relational Calculus both have equivalent expressive power. The main difference between them is just that Relational Algebra specify how to retrieve data and Relational Calculus defines what data is to be retrieved

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

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

In addition to the fundamental operations, there are several other operations—namely, set intersection, natural join, division, and assignment.We will define these operations in terms of the fundamental operations.

2.4.3 Select Operation (σ)

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

Notation:

$$\sigma_p(r) \tag{2.3}$$

Where σ stands for selection predicate and r stands for relation. p is prepositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like $=, \neq, \geq, <, >, \leq$.

To select those tuples of the loan relation where the branch is "Perryridge," we write

$$\sigma_{branch\text{--}name} = Perryridge(loan)$$

We can find all tuples in which the amount lent is more than $ 1200 by writing

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

To find those tuples pertaining to loans of more than $ 1200 made by the Perryridge branch, we write

$$\sigma_{branch-name} = Perryridge \wedge amount > 1200(loan)$$

2.4.4 Project Operation (Π)

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

Notation:

$$\Pi A_1, A_2, A_n(r) \quad (2.4)$$

Where A_1, A_2 , A_n are attribute names of relation r. Duplicate rows are automatically eliminated, as relation is a set.

To list all loan numbers and the amount of the loan:

$$\Pi_{loan-number,amount}(loan) \quad (2.5)$$

Composition of Relational Operations

To "Find those customers who live in Harrison." We write:

$$\Pi_{customer-name}(\sigma_{customer-city} = Harrison(customer)) \quad (2.6)$$

We can define the three operations UNION, INTERSECTION, and SET DIFFERENCE on two union-compatible relations R and S as follows:

- **UNION:**
  The result of this operation, denoted by

  $$R \cup S$$

  , is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

- **INTERSECTION:**
  The result of this operation, denoted by

  $$R \cap S$$

  , is a relation that includes all tuples that are in both R and S.

- **SET DIFFERENCE (or MINUS):**
  The result of this operation, denoted by

  $$R - S$$

  , is a relation that includes all tuples that are in R but not in S.

2.4.5 The Union Operation

Consider a query to find the names of all bank customers who have either an account or a loan or both.

$$\Pi_{customer\text{-}name}(borrower) \cup \Pi_{customer\text{-}name}(depositor) \quad (2.7)$$

For a union operation $R \cup S$ to be valid, we require that two conditions hold:

1. The relations r and s must be of the same arity. That is, they must have the same number of attributes.
2. The domains of the i^{th} attribute of r and the i^{th} attribute of s must be the same, for all i.

2.4.6 The Set Difference Operation

Notation :

$$r - s$$

The set-difference operation, denoted by , allows us to find tuples that are in one relation but are not in another. The expression r-s produces a relation containing those tuples in r but not in s.

To find all customers of the bank who have an account but not a loan

$$\Pi_{customer\text{-}name}(depositor)\Pi_{customer\text{-}name}(borrower) \quad (2.8)$$

As with the union operation, we must ensure that set differences are taken between compatible relations.

- Tuple relational calculus which was originally proposed by Codd in the year 1972 and
- Domain relational calculus which was proposed by Lacroix and Pirotte in the year 1977

  In first-order logic or predicate calculus, a predicate is a truth-valued function with arguments. When we replace with values for the arguments, the function yields an expression, called a proposition, which will be either true or false.

Example:
For example, steps involved in listing all the employees who attend the 'Networking' Course would be:

```
SELECT the tuples from COURSE
relation with COURSENAME = 'NETWORKING'
PROJECT the COURSE _ID from above result
```

Listing 2.10: Example

SELECT the tuples from EMP relation with COURSE*IDresultedabove*.

Tuple Relational Calculus

In the tuple relational calculus, you will have to find tuples for which a predicate is true. The calculus is dependent on the use of tuple variables. A tuple variable is a variable that 'ranges over' a named relation: i.e., a variable whose only permitted values are tuples of the relation. **Domain Relational calculus:**

In the tuple relational calculus, you have use variables that have a series of tuples in a relation. In the domain relational calculus, you will also use variables, but in this case, the variables take their values from domains of attributes rather than tuples of relations. A domain relational calculus expression has the following general format:

$$d_1, d_2, ..., d_n | F(d_1, d_2, ..., d_m) m \geq n \qquad (2.9)$$

where $d_1, d_2, ..., d_n, ..., dm$ stand for domain variables and $F(d_1, d_2, ..., d_m)$ stands for a formula composed of atoms.

Chapter 3

Structured Query Language (SQL)

3.1 Introduction to SQL

SQL is a database computer language designed for the retrieval and management ofdata in a relational database. SQL stands for Structured Query Language. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQLas their standard database language. Originally, SQL was called SEQUEL (Structured English QUEry Language). SQL uses the terms table, row, and column for the formal relational model terms relation, tuple, and attribute, respectively.

SQL uses the concept of a catalog—a named collection of schemas in an SQL environment. An SQL environment is basically an installation of an SQL-compliant RDBMS on a computer system. A catalog always contains a special schema called INFORMATION_SCHEMA, which provides information on all the schemas in the catalog andall the element descriptors in these schemas.

SQL Constraints

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifieseach row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified

SQL Commands

- DDL: Data Definition Language
- DML: Data Manipulation Language
- DCL: Data Control Language
- TCL:Transaction Control Language

3.2 Data Definition Language (DDL):

The SQL DDL allows specification of not only a set of relations, but also information about each relation, including:

- The schema for each relation.
- The types of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.

Basic data types used are,

char	A fixed-length character string with user-specified length n.The fullform, character, can be used instead.
varchar	A variable-length character string with user-specified maximumlength n. The full form, character varying, isequivalent.
int	An integer (a finite subset of the integers that is machine dependent).Thefull form, integer, is equivalent.
float	A floating-point number, with precision of at least n digits.

Main Commands

CREATE	Creates a new table, a view of a table, or otherobject in thedatabase.
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or otherobjects inthe database.
Rename	Rename a table or its attribute.
Truncate	Operation that is used to mark the extents of atable fordeallocation (empty for reuse)

Create

- To create database,

  CREATE DATABASE database_name

- To create Table Statement is used to create tables to store data. Integrity Constraintscan alsobe defined for the columns while creating the table.

CREATE TABLE table name(Attribute1 datatype(No.),...An datatype(No));
CREATE TABLE employee (id number(5),name char(20),dept char(10));

- To create table constraint

CREATE TABLE *table_name* (A*1 datatype constraint*, *A2 datatype constraint*, *A3 datatypeconstraint*,);

-primary key: The primary key attributes are required to be nonnull and unique.

CREATE TABLE Persons (PID int(10) NOT NULL PRIMARY KEY, LastName varchar(25),FirstName varchar(25));

Or

CREATE TABLE Persons (PID int (10) NOT NULL, LastName varchar(25),FirstNamevarchar(25), primary key(id));

-Foreign Key

CREATE TABLE Orders (OrderID int NOT NULL PRIMARY KEY, OrderNumber int NOTNULL, PersonID int, FOREIGN KEY (PID) REFERENCES Persons(PID));

ALTER:

- To add column

  ALTER TABLE table_name ADD column_name datatype;
- To delete column

  ALTER TABLE table_name Drop column column_name;

DROP:

DROP DATABASE database_name;
DROP TABLE table_name;

RENAME:

- To rename a table

  RENAME TABLE tbl_name TO new_tbl_name**;**
- To rename a column

  ALTER TABLE table_name Rename column old column name to new name;

3.3 Data Manipulation Language(DML):

The SQL commands that deals with the manipulation of data present in database belongto DML or Data Manipulation Language and this includes most of the SQL statements.

SELECT	used to retrieve data from the a database.
UPDATE	used to update existing data
INSERT	used to insert data into a table.
DELETE	used to delete records

INSERT

INSERT INTO *tablename* (*column1*, *column2*, ..)VALUES (*value1*, *value2*,.. .);

OR

INSERT INTO *table_name* VALUES (*value1*, *value2*, *value3*, ...);

SELECT

- To select a entire table

SELECT * FROM table_name;

UPDATE

SELECT *column1*, *column2,* ...FROM *table_name* WHERE *condition*;

UPDATE table_name SET column1 = value1, column2 = value2,... WHERE condition;

DELETE

- To delete all rows

DELETE FROM *table_name*;

- To delete specific row

DELETE FROM table_name WHERE condition;

3.4 Data Control Language (DCL):

DCL mainly deals with the rights, permissions and other controls of the database system

GRANT	gives user's access privileges to database.
REVOKE	withdraw user's access privileges given by using the GRANT command.

GRANT

GRANT privilege_name ON Table_name TO user_name;

Eg. GRANT SELECT ON employee TO user1

REVOKE

REVOKE privilege_name ON Table_name FROM user_name;

Eg. REVOKE SELECT ON employee FROM user1;

3.5 Transaction Control Language (TCL):

Transaction Control Language (TCL) commands are used to manage transactions in the database. Theseare used to manage the changes made to the data in a table by DML statements.

COMMIT	command is used to permanently save any transaction into the database.
ROLLBACK	command to rollback changes
SAVEPOINT	command is used to temporarily save a transaction so that you can rollbackto that point whenever required.

COMMIT:

Syntax- COMMIT;

ROLLBACK:

- The ROLLBACK command to rollback those changes, if they were not committed usingtheCOMMIT command

  Rollback;

- The command restores the database to last committed state by using

  SAVEPOINTcommand.ROLLBACK TO savepoint_name;

SAVEPOINT

SAVEPOINT savepoint_name;

3.6 SELECT Queries

The basic structure of an SQL query consists of three clauses: select, from, and where. The query takes as its input the relations listed in the from clause, operates on them as specified in the where and select clauses, and then producesa relation as the result.

3.6.1 Queries on a Single Relation

Let us consider the below table Faculty and DEPT,

FID	FNAME	DEPTID	SALARY
1	JISY	1	35000
2	SANTHY	1	30000
3	SWETHA	2	25000

DEPTID	DEPTNAME	Block
1	CS	New
2	EC	New
3	EE	old

Queries on a Single Relation

Let us consider a simple query using our Faculty table, “Find the names of all instructors.

select FNAME from Faculty;

The result is a relation consisting of a single attribute. If want to force the elimination of duplicates, we insert the keyword **distinct** after select. We can rewrite the preceding query as:

FID	FNAME	DEPTNAME
1	JISY	1
2	SANTHY	1
3	SWETHA	2

select distinct DEPTNAME from FACULTY;

DEPT
CS
EC

SQL allows us to use the keyword **all** to specify explicitly that duplicates are not removed:

select all DEPTNAME from DEPT;

The select clause may also contain arithmetic expressions involving the operators +, –, *, and /operating on constants or attributes of tuples. For example,the query returns a relation that is the same as the Faculty relation, except that the attribute salary is multiplied by 1.1.

select FID , FNAME, SALARY * 1.1 from Faculty;

SQL allows the use of the logical connectives and, or, and not in the where clause. The operands of the logical connectives can be expressions involving the comparison operators
<, <=, >, >=, =, and <>.

select FNAME from Faculty where SALARY>30000;

3.6.2 Queries on Multiple Relations

Consider two tables,

CID	NAME	Addr
1	Manju	abc
2	Jisy	cde
3	Vishnu	efg
4	Meera	hij

OID	CID	AMOUNT
1	2	100
2	1	250
3	4	300
4	3	400

- Retrieve CID, Address and amount from relation CUSTOMER and ORDERwhosename= jisy

SELECT CUSTOMER.CID, Addr, AMOUNT FROM CUSTOMER, ORDER WHERE CUSTOMERS.CID = ORDERS.CID and NAME='Jisy';

- Retrieve customer id, name, Address and amount from relation CUSTOMERandORDER

  SELECT CUSTOMER.CID, NAME, Addr, AMOUNT FROM CUSTOMER, ORDERWHERE CUSTOMERS.CID = ORDERS.CID;

3.6.3 Join

Select CID, NAME, Addr, AMOUNT from CUSTOMER **Natural join** ORDER

Select CID, NAME, Addr, AMOUNT from CUSTOMER **Inner join** ORDER onCUSTOMER.CID = ORDER.CID;

SQL aliases/ correlation name/ tuple variable.

SQL aliases are used to give a table, or a column in a table, a temporary name.

- To rename column,

  Select old column name as new name from table name; Eg. Select CID as CustomerID , Name from CUSTOMER;

- To rename table

  Select Name from Customer as Cust where CID=1;

  SELECT C.CID, NAME, Addr, AMOUNT FROM CUSTOMER as C, ORDERWHERE C.CID = ORDERS.CID;

3.6.4 String Operations

SQL specifies strings by enclosing them in single quotes, for example, 'Computer'. The SQL standard specifies that the equality operation on strings is case sensitive; as a result the expression " 'computer' = 'Computer' " evaluates to false.

SQL also permits a variety of functions on character strings, such as concatenating (using " ||"), extracting substrings, finding the length of strings, converting strings to uppercase (using the function upper(s) where s is a string) and lowercase (using the function lower(s)), removing spaces at the end of the string (using trim(s)). Pattern matching can be performed on strings, using the operator **like**. We describe patterns by using two special characters:

• Percent (%): multiple character
• Underscore (_): single character.

SELECT *column1, column2,* ...FROM *table_name* WHERE *columnN* LIKE *pattern*; SELECT * FROM CUSTOMER WHERE Name LIKE 'a%';

WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in anyposition
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in thesecond position
WHERE CustomerName LIKE 'a_%_%'	Finds any values that start with "a" and areatleast 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" andends with "o"

3.6.5 SQL ORDER BY

The ORDER BY statement in sql is used to sort the fetched data in either ascending or descending according to one or more columns. By default ORDER BY sorts the data in ascending order.

SELECT *column1*, *column2,* ...FROM *table_name* ORDER BY *column1,column2,* ... ASC/DESC;

Eg. SELECT NAME FROM CUSTOMER ORDER BY Name DESC;

NAME
Vishnu
Manju
Jisy

3.6.6 SQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates. The BETWEEN operator is inclusive: begin and end values are included.

SELECT *column_name(s)* FROM *table_name* WHERE *column_name* BETWEEN
value1 AND *value2;*
SELECT * FROM ORDER WHERE AMOUNT BETWEEN 100 AND 350;

OID	CID	AMOUNT
1	2	100
2	1	250
3	4	300

3.6.7 Set Operations

The SQL operations union, intersect, and except operate on relations and correspond to the mathematical set-theory operations ∪, ∩, and –. Consider two tables First and Second.

ID	Name
1	JISY
2	SANTHY

ID	Name
3	SWETHA
2	SANTHY

UNION Operation:is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its resultset. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.

SELECT * FROM First UNION SELECT * FROM Second;

ID	Name
1	JISY
2	SANTHY
3	SWETHA

UNION ALL: This operation is similar to Union. But it also shows the duplicate rows.

SELECT * FROM First UNION ALL SELECT * FROM Second;

ID	Name
1	JISY
2	SANTHY
3	SWETHA
2	SANTHY

INTERSECT: Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of **Intersect** the numberof columnsand datatype must be same

SELECT * FROM First INTERSECT SELECT * FROM Second;

ID	Name
2	SANTHY

Minus/ Except: It combines the result of two SELECT statements. Minus operator is used todisplaythe rows which are present in the first query but absent in the second query.

SELECT * FROM First Except SELECT * FROM Second;

ID	Name
1	JISY

SELECT * FROM Second MINUS SELECT * FROM First ;

ID	Name
3	SWETHA

3.6.8 Aggregate Functions

Aggregate functions are functions that take a collection (a set or multiset) of values as input andreturn asingle value.

Basic Aggregation

SQL offers five built-in aggregate functions:

- Average: avg
- Minimum: min
- Maximum: max
- Total: sum
- Count: count

Select Aggregate fn(column name) from table_name where condition;
Select Max(Mark) from Stud ;

Stud

RollNo.	Name	Mark	Dept
1	A	40	cs
2	B	36	cs
3	C	28	ec
4	B	30	ec
5	F	46	ee
6	G	34	cs

AVG(): SELECT AVG(*column_name*) FROM *table_name* WHERE *condition*;Select avg(Mark) from Stud;

COUNT(): The aggregate function count used to count the number of tuples in a relation.
Select Count(*) from Stud;

Count(*)
6

Select Count(*) from Stud where Name='B';

Count(*)
2

Select Count (Distinct Name) from Stud;

Name
A
B
C
F
G

MIN(): The MIN() function returns the smallest value of the columns.
Select Min(Mark) from Stud ;

Min
28

Max(): The MAX() function returns the largest value of the selected column.

Max
46

Sum(): The SUM() function returns the total sum of a numeric column.
Select sum(Mark) from Stud ;
Select sum(M1+M2) from Stud ; (also possible)

Aggregation with Group by: The GROUP BY statement is often used with aggregatefunctions.
Select Aggregate fn(column name) from table_name **group by** column name;SelectDept, Count(RollNo) from Stud Group By Dept;

Dept	Count
cs	3
ec	2
ee	1

Group by using the HAVING clause: Grouping data with certain condition.

SELECT column_name(s) FROM table_name WHERE condition **GROUP BY** column name(s) **HAVING** condition

Select Dept, Count(RollNo) from Stud Group By Dept Having Mark>35;

Dept	Count
cs	2
ec	0
ee	1

Aggregation with Null and Boolean Values

In general, aggregate functions treat nulls according to the following rule: All aggregate functions except count (*) ignore null values in their input collection. As a result of null values being ignored, the collection of values may be empty. The count of an empty collection is defined to be 0, and all other aggregate operations return a value of null when applied on an empty collection. A Boolean data type that can take only values true, false, and unknown.

3.7 Views (Virtual Table)

A view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either haveall the rows of a table or specific rows based on certain condition. Views are a logical virtual table created by "select query" but the result is not stored anywhere in the disk and every time we need to fire the query when we need data, so alwayswe getupdated or latest data from original tables.

SD

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

SM

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Creating Views

A View can be created from a single table or multiple tables.

CREATE VIEW view_name AS SELECT column1, column2..... FROM table_name WHEREcondition;

- **Creating View from a single table:**
  CREATE VIEW Details AS SELECT NAME, ADDRESS FROM SD WHERE S_ID < 5;

To see the data in the View, we can query the view in the same manner as we query a table.
SELECT * FROM Details;

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

- **Creating View from multiple tables**:CREATE VIEW Marks AS
  SELECT SD.NAME, SD.ADDRESS, SM.MARKS FROM SD, SM WHERE SD.NAME =SM.NAME;

NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

UPDATING VIEWS

A view can be updated under certain conditions which are given below –

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.

We can use the CREATE OR REPLACE VIEW statement to add or remove fields from a view.

CREATE OR REPLACE VIEW view_name AS SELECT column1,coulmn2,..FROM table_name

WHERE condition;

For example, if we want to update the view **Marks** and add the field AGE to this View from **SM** Table, we can do this as:

CREATE OR REPLACE VIEW Marks AS SELECT SD.NAME, SD.ADDRESS, SM.MARKS,SM.AGE FROM SD,SM WHERE SD.NAME = SM.NAME;

NAME	ADDRESS	MARKS	AGE
Harsh	Kolkata	90	19
Pratik	Delhi	80	19
Dhanraj	Bihar	95	21
Ram	Rajasthan	85	18

Inserting a row

INSERT INTO View name(C1,C2...Cn) VALUES(V1,V2....Vn);

INSERT INTO Details(NAME, ADDRESS) VALUES("Suresh","Gurgaon");

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar
Suresh	Gurgaon

DELETING Rows

Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.

DELETE FROM view_name WHERE condition;

DELETE FROM Details WHERE NAME="Suresh";

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

DELETING VIEWS

We can delete or drop a View using the DROP statement.

DROP VIEW view_name;

Materialized View

Materialized views are also the logical view of our data-driven by the select query but the resultof thequery will get stored in the table or disk.

Materialized View vs View

View	Materialized View
Views query result is not stored in the disk ordatabase	Materialized view allow to store the queryresult in disk or table.
when we create a view using any table, rowidof view is same as the original table	Materialized view rowid is different.
View we always get latest data	Materialized view we need to refresh the view for getting latest data.
Performance of View is less than Materialized view.	Performance of Materialized View is higherthan view.

3.8 Joining Databases

We can retrieve data from more than one tables using the JOIN statement. There are mainly 4 different types ofJOINS in SQL server. We will learn all JOINS in SQL server with examples:

- INNER JOIN/simple join
- LEFT OUTER JOIN/LEFT JOIN
- RIGHT OUTER JOIN/RIGHT JOIN
- FULL OUTER JOIN

INNER JOIN

This type of SQL server JOIN returns rows from all tables in which the join condition is true. It takes thefollowing syntax:

```
SELECT columns
FROM table_1
INNER JOIN
table_2
ON table_1.column = table_2.column;
```

We will use the following two tables to demonstrate this:

Students Table:

	admission	firstName	lastName	age
1	3420	Nicholas	Samuel	14
2	3380	Joel	John	15
3	3410	Japheth	Becky	16
4	3398	George	Joshua	14
5	3386	John	Lucky	15
6	3403	Simon	Dan	13
7	3400	Calton	Becham	16

Fee table:

	admission	course	amount_paid
1	3380	Electrical	20000
2	3420	ICT	15000
3	3398	Commerce	13000
4	3410	HR	12000

The following command demonstrates an INNER JOIN in SQL server with example:

SELECT Students.admission, Students.firstName, Students.lastName,

Fee.amount_paidFROM Students

INNER JOIN Fee

ON Students.admission =

Fee.admissionThe command returns the

following:

	admission	firstName	lastName	amount_paid
1	3420	Nicholas	Samuel	15000
2	3380	Joel	John	20000
3	3410	Japheth	Becky	12000
4	3398	George	Joshua	13000

We can tell the students who have paid their fee. We used the column with common values in both tables, whichis the admission column.

LEFT OUTER JOIN

This type of join will return all rows from the left-hand table plus records in the right-hand table with matchingvalues. For example:

SELECT Students.admission, Students.firstName, Students.lastName,

Fee.amount_paidFROM Students

LEFT OUTER JOIN Fee

ON Students.admission =

Fee.admissionThe code returns the

following:

	admission	firstName	lastName	amount_paid
1	3420	Nicholas	Samuel	15000
2	3380	Joel	John	20000
3	3410	Japheth	Becky	12000
4	3398	George	Joshua	13000
5	3386	John	Lucky	NULL
6	3403	Simon	Dan	NULL
7	3400	Calton	Becham	NULL

The records without matching values are replaced with NULLs in the respective columns.

RIGHT OUTER JOIN

This type of join returns all rows from the right-hand table and only those with matching values in the left-handtable. For example:

```
SELECT Students.admission, Students.firstName, Students.lastName,
Fee.amount_paidFROM Students
RIGHT OUTER JOIN Fee
ON Students.admission = Fee.admission
```

The statement for OUTER JOINS SQL server returns the following:

	admission	firstName	lastName	amount_paid
1	3380	Joel	John	20000
2	3420	Nicholas	Samuel	15000
3	3398	George	Joshua	13000
4	3410	Japheth	Becky	12000

The reason for the above output is that all rows in the Fee table are available in the Students table when matchedon the admission column.

FULL OUTER JOIN

This type of join returns all rows from both tables with NULL values where the JOIN condition is not true. Forexample:

```
SELECT Students.admission, Students.firstName, Students.lastName,
Fee.amount_paidFROM Students
FULL OUTER JOIN Fee
ON Students.admission = Fee.admission
```

The code returns the following result for FULL OUTER JOINS queries in SQL:

	admission	firstName	lastName	amount_paid
1	3420	Nicholas	Samuel	15000
2	3380	Joel	John	20000
3	3410	Japheth	Becky	12000
4	3398	George	Joshua	13000
5	3386	John	Lucky	NULL
6	3403	Simon	Dan	NULL
7	3400	Calton	Becham	NULL

3.9 Sub Queries and Correlated Queries

A sub-query or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause. A subquery is a SELECT statement that is nested within another SELECT statement and which return intermediate results. The result of inner query is used in execution of outer query.

Stud

SID	Name	Mark	Dept
1	A	40	cs
2	B	36	cs
3	C	28	ec
4	B	30	ec
5	F	46	ee
6	G	34	cs

Course

CID	Cname
c1	DBMS
c2	DS
c3	CP

Scourse

SID	CID
1	c1
1	c2
2	c3
3	c2
4	c3

Independent Nested Queries: In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.

IN (Set Membership)

The IN connective for set membership, where the set is a collection of values producedby aselect clause. The NOT IN connective for the absence of set membership.

SELECT column-names FROM table-name1 WHERE value IN (SELECT column-nameFROMtable-name2 WHERE condition)

Q1. If we want to find out SID who are enrolled in Cname 'DS' or 'DBMS'.

From Course table, we can find out CID for Cname 'DS' or DBMS' and we can use these CIDsforfinding SIDs from Scourse TABLE.

STEP 1: Finding CID for Cname ='DS' or 'DBMS'

Select CID from Course where Cname = 'DS' or Cname = 'DBMS';

STEP 2: Using CID of step 1 for finding SID

Select SID from Scourse where CID **IN** (Select CID from Course where Cname = 'DS'orCname = 'DBMS');

Q2. Find out names of STUDENTs who have either enrolled in 'DS' or 'DBMS', it can be doneas:

Select Name from Stud where SID **IN** (Select SID from Scourse where CID **IN** (Select CIDfrom Course where Cname = 'DS' or Cname = 'DBMS'));

Q3. If we want to find out SIDs of STUDENTs who have neither enrolled in 'DSA' nor in 'DBMS',it canbe done as:

Select Name from Stud where SID **NOT IN** (Select SID from Scourse where CID **IN** (Select CID from Course where Cname = 'DS' or Cname = 'DBMS'));

Test for Empty Relations

SQL includes a feature for testing whether a subquery has any tuples in its result.

The **exists** construct returns the value true if the argument subquery is nonempty. We can test for the nonexistence of tuples ina subquery by using the **not exists** construct.

Q1. If we want to find out NAME of Student who are enrolled in CID 'C1'

Select NAME from Stud where **EXISTS**(select * from Scourse where Stud.SID=Scourse.SID and Scourse.CID='C1');

Correlated Query: With a normal nested subquery, the inner SELECT query runs first and executes once, returning values to be used by the main query. A **correlated subquery** is a subquery that uses values from the outer query.
Eg.SELECT employee_number, name FROM employees emp WHERE salary > (SELECT AVG(salary) FROM employees WHERE department = emp.department);

Test for the Absence of Duplicate Tuples

Unique constraint in SQL is used to check whether the sub query has duplicate tuples in its result. Unique construct returns true only if the sub query has no duplicate tuples, else it return false. We can test for the existence of duplicate tuples in a subquery by using the not unique construct.

Q1. Find course ID who enrolled in at least one course?

SELECT Course.CID FROM Course WHERE UNIQUE (SELECT CID FROM Scourse where Scourse.CID=Course.CID);

ALL

The ALL operator returns TRUE if all of the subqueries values meet the condition.

SELECT column-names FROM table-name WHERE column-name operator ALL(SELECT column-name FROM table-name WHERE condition)

Q1. Returns the names, Rollno of students whose mark is greater than the mark of all the students in department ec:

SELECT Name , SID FROM Stud WHERE Mark > ALL (SELECT Mark FROM Stud WHERE Dept =ec);

Chapter 4

Schema Refinement

4.1: Problems caused by redundancy

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

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

Redundancy refers to repetition of same data or duplicate copies of same data stored indifferent locations.

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

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

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

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

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

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

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

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

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

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

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

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

7k
7k
Costly Operation
More IO Cost

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

InsertionAnomalies: New course is introduced C4, But no student is there who is having C4subject.

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

NULL	NULL	CA	DB	12k

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

Therefore,

xx	xx	CA	DB	12k

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

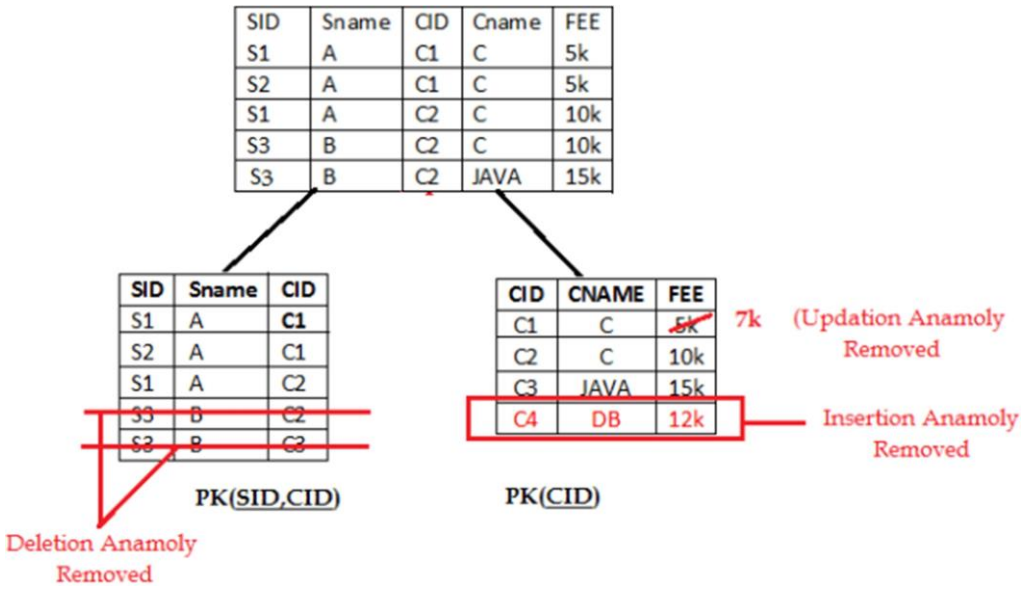
Deletion Anomaly :

Deletion of S3 student cause the deletion of course.

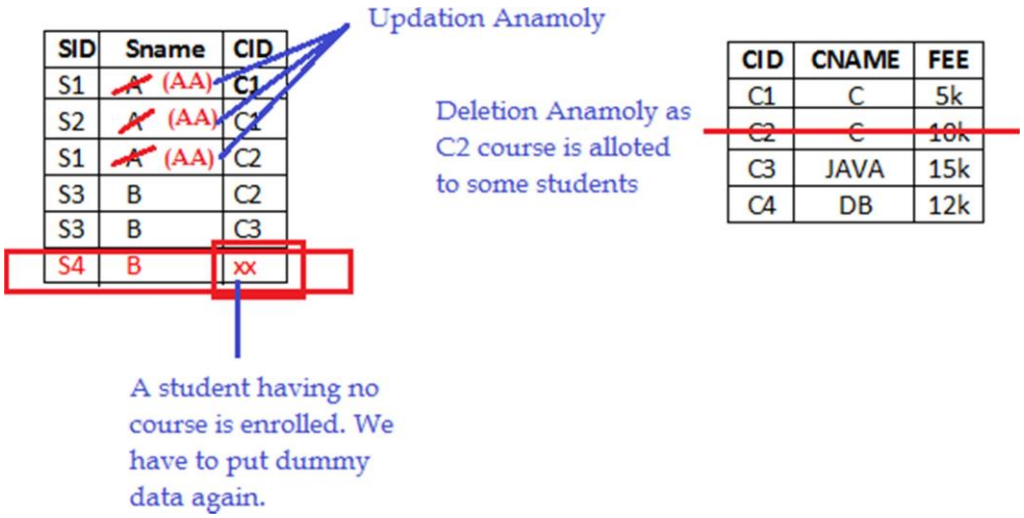
Because of deletion of some data forced to delete some other useful data.

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

Solutions To Ano
malies : Decomposition of Tables – Schema Refinement

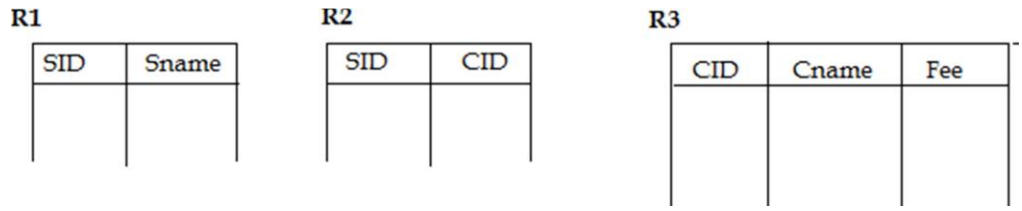


There are some Anomalies in this again –



What is the Solution ??

Solution : decomposing into relations as shown below



4.2 Decomposition

☐**TO AVOID REDUNDANCY** and problems due to redundancy, we use refinement techniquecalled **DECOMPOSITION.**
Process of decomposing a larger relation into smaller relations.

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

Functional dependencies:

☐Functional dependency is a relationship that exists when one attribute uniquely determinesanother attribute.

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

☐A functional dependency A☐B in a relation holds true if two tuples having the same value ofattribute A also have the same value of attribute B

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

4.4 Reasoning about functional dependencies

Armstrong Axioms:

Armstrong axioms define the set of rules for reasoning about functional dependencies andalso to infer all the functional dependencies on a relational database.

Various axioms rules or inference rules:

Primary axioms:

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

secondary or derived axioms:

Rule 1	**Union** If A holds B and A holds C, then A holds BC. If{A → B} and {A → C}, then {A → BC}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If{A → BC} and {A → B}, then {A → C}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If{A → B} and {BC → D}, then {AC → D}

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

NOTE:

To find attribute closure of an attribute set-

1) Add elements of attribute set to the result set.

2) Recursively adds elements to the result set which can be functionally determined from theelements of result set.

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

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

Steps:
            1. X+ = X                              //initialize X+ to X
            2. For each FD : Y -> Z in F Do
               If Y ⊆ X+ Then                      //If Y is contained in X+
               X+ =  X+ ∪ Z                        //add Z to X+
               End If
               End For
            3. Return  X+                         //Return closure of X

Output : Closure  X+ of X under F
```

Types of functional dependencies:

1)**Trivial functional dependency:-**If X☐Y is a functional dependency where Y subset X, thesetype of FD's called as trivial functional dependency.

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

3)**Completely non-trivial functional dependency:**-If X☐Y and X∩Y=Φ(null) then it is called completely non-trivial functional dependency.

Prime and non-prime attributes

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

Candidate Key:
Candidate Key is minimal set of attributes of a relation which can be used to identify a tupleuniquely.
Consider student table: student(sno, sname,sphone,age) we can take **sno** as candidate key.

We can have more than 1 candidate key in a table.
Ttypes of candidate keys:
1. simple(having only one attribute)
2. composite(having multiple attributes as candidate key)

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

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

Consider student table: student (sno, sname,sphone,age)we can take sno, (sno, sname) as super key

Finding candidate keys problems:

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

AB→CD, C→A, D→A

Solution:

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

C→A replace A by c

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

D→B replace B by D

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

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

AB, BC, CD, AD are candidate keys.

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

A→BC,CD→E,B→D, E→A

Solution:

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

E→A so replace A by E

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

CD→E replace E by CD

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

B→D

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

A, E, CD, BC are candidate keys

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

Solution :

$(AB)^+$: {ABCDEF} ⇒ Super Key
$(A)^+$: {A} ⇒ Not Super Key
$(B)^+$: {B} ⇒ Not Super Key
Prime Attributes : {A,B}
(AB) → Candidate Key
↓ (as F → B)
$(AF)^+$: {AFBCDE}
$(A)^+$: {A} ⇒ Not Super key
$(F)^+$: {FB} ⇒ Not Super Key
(AF) → Candidate Key
↓
$(AE)^+$: {AEFBCD}
$(A)^+$: {A} ⇒ Not Super key
$(E)^+$: {EFB} ⇒ Not Super key
(AE) → Candidate Key
↓
$(AD)^+$: {ADEFBC}
$(A)^+$: {A} ⇒ Not Super key
$(D)^+$: {DEFB} ⇒ Not Super key
(AD) → Candidate Key
↓
$(AC)^+$: {ACDEFB}
$(A)^+$: {A} ⇒ Not Super Key
$(C)^+$: {DCEFB} ⇒ Not Super Key
⇒ Candidate Keys {AB, AF, AE, AD, AC}
⇒ Prime Attributes {A,B,C,D,E,F}
⇒ Non Prime Attributes {}

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

Solution :

$(AB)^+$: {A B C D E F}
$(A)^+$: {A}
$(B)^+$: {B}
(AB) ⇒ (AC), $(AC)^+$: {ABCDEF}
$(C)^+$: {DECBF}
⇒ Candidate Keys {AB, AC}
⇒ Prime Attributes {A,B,C}
⇒ Non Prime Attributes {D,E,F}

Normalization:

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

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

1) First normal form(1NF)

2)Second normal form(2NF)

3)Third normal form(3NF)

4) Boyce coded normal form(BCNF)

5)Forth normal form(4NF)

6) Fifth normal form (5NF)

7) Sixth normal form(6NF)

8)Domain key normal form.

Normal Forms

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

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

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

Properties of normal forms and their decompositions

4.5 First Normal Form (1NF)

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

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

EMPLOYEE table:

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

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

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

4.6 Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

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

TEACHER table

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

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

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

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

4.7 Third Normal Form

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

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

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

Example:

EMPLOYEE_DETAIL table:

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

Super key in the table above:

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

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

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

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

EMPLOYEE table:

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

EMPLOYEE_ZIP table:

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

4.8 Boyce-Codd Normal Form

It is an extension of third normal form where in a functionaldependency X→A , X must be a super key.

A relation is in BCNF if in every non-trivial functional dependency X –> Y, X is a super key.

BCNF (Boyce Codd Normal Form) is the advanced version of 3NF. A table is in BCNF if every functional dependency X->Y, X is the super key of the table. For BCNF, the table should be in 3NF, and for every FD. LHS is super key.

Example

Consider a relation R with attributes (student, subject, teacher).

Student	Teacher	Subject
Jhansi	P.Naresh	Database
jhansi	K.Das	C
subbu	P.Naresh	Database
subbu	R.Prasad	C

F: { (student, Teacher) -> subject

(student, subject) -> Teacher

Teacher -> subject}

Candidate keys are (student, teacher) and (student, subject).

The above relation is in 3NF [since there is no transitive dependency]. A relation R is in BCNF if for every non-trivial FD X->Y, X must be a key.

The above relation is not in BCNF, because in the FD (teacher->subject), teacher is not a key. This relation suffers with anomalies –

For example, if we try to delete the student Subbu, we will lose the information that R. Prasad teaches C. These difficulties are caused by the fact the teacher is determinant but not a candidate key.

Decomposition for BCNF

Teacher-> subject violates BCNF [since teacher is not a candidate key].

If X->Y violates BCNF then divide R into R1(X, Y) and R2(R-Y).

So R is divided into two relations R1(Teacher, subject) and R2(student, Teacher).

R1

Teacher	Subject
P.Naresh	Database
K.DAS	C
R.Prasad	C

R2

Student	Teacher
Jhansi	P.Naresh
Jhansi	K.Das
Subbu	P.Naresh
Subbu	R.Prasad

All the anomalies which were present in R, now removed in the above two relations.

4.9 Fourth Normal Form

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

STUDENT

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

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

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

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

STUDENT_COURSE

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

STUDENT_HOBBY

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

Fifth Normal Form

Fifth Normal Form is related to join dependencies.

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

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

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

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

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

5 Domain key normal form: A domain key normal form keeps a constraint that every constrainton the relation is a logical sequence of definition of keys and domains.

6 Sixth normal form: A relation is said to be in sixth normal form such that the relation R shouldnot contain any non-trivial join dependencies.

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

Key Points related to normal forms –

BCNF is free from redundancy.

If a relation is in BCNF, then 3NF is also also satisfied.

If all attributes of relation are prime attribute, then the relation is always in 3NF.

A relation in a Relational Database is always and at least in 1NF form.

Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.

If a Relation has only singleton candidate keys (i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).

Sometimes going for BCNF form may not preserve functional dependency. In that case gofor BCNF only if the lost FD(s) is not required, else normalize till 3NF only.

There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

Problems on normal forms:
Problem 1:

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

Solution:
Important points for solving above type of question:

1) It is always a good idea to start checking from BCNF, then 3 NF and so on.
2)
3) If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, ABC –> D is in BCNF (Note that ABC is a super key), so no need to check this dependency for lower normal forms.
4)
Candidate keys in given relation is {ABC, BCD}

BCNF: ABC -> D is in BCNF. Let us check CD -> AE, CD is not a super key so this dependency is notin BCNF. So, R is not in BCNF.

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

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

problem 2:

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

withFD set as {BC->D,AC->BE,B->E}

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

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

Step 3: The relation R is in 1st normal form as a relational DBMS does not allow multi-valued orcomposite attribute.
The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not proper subsetof candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).
The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rdnormal for, either LHS of an FD should be super key or RHS should be prime attribute. So the highest normal form of relation will be 2nd Normal form.

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

Rules of decomposition:

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

1) Union of two smaller subsets of attributes gives all attributes

of 'R'.R1(attributes)UR2(attributes)=R(attributes)

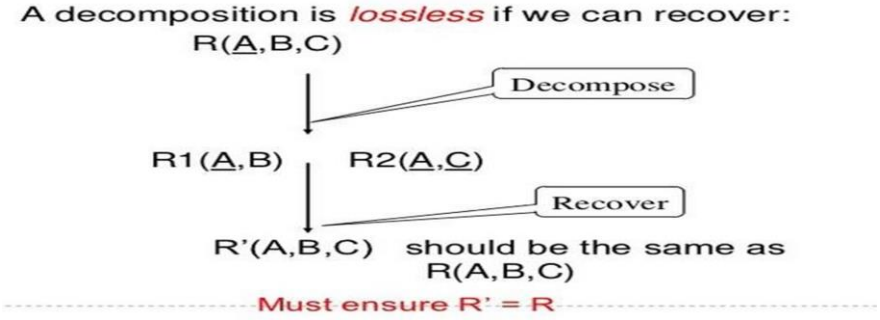
2) Both relations interaction should not give null

value.R1(attributes)∩R2(attributes)!=null

3) Both relations interaction should give key attribute.

R1(attribute)∩R2(attribute)=R(key ttribute)

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



tables.

Example 2 for loseless decomposition:

Lossless Decomposition (example)

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

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \rightarrow B;\ C \rightarrow B$

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

=

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

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

Access	100

Access	DB

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

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

Example 1:

Lossy Decomposition (example)

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

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

$A \rightarrow B;\ C \rightarrow B$

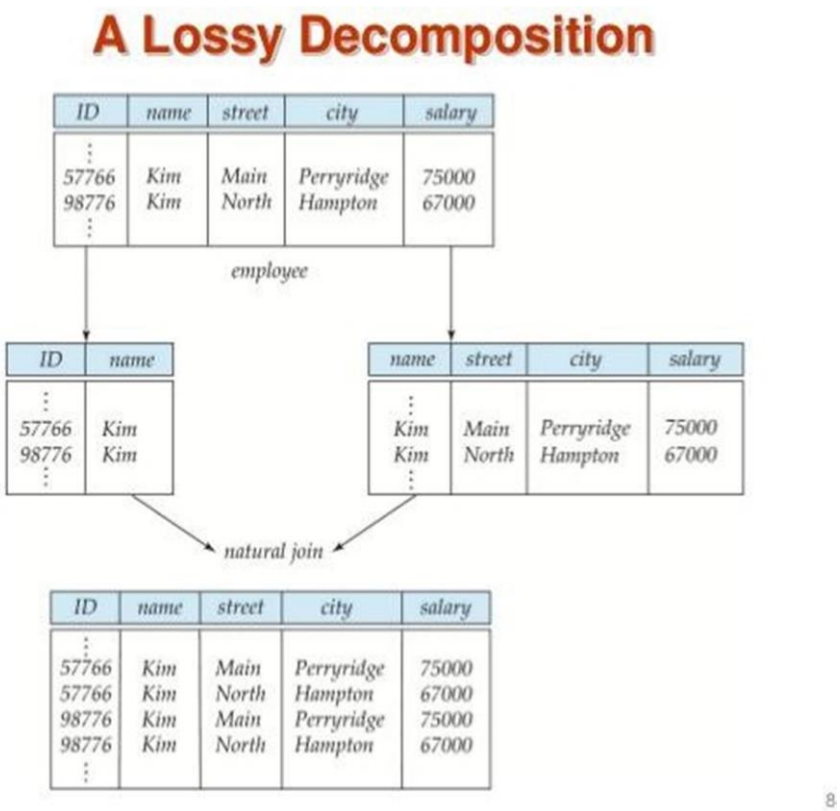
A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

=

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

Example 2:



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

generated.so it is lossy join decomposition.

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

Dependency Preservation

A Decomposition D = { R1, R2, R3....Rn } of R is dependency preserving wrt a set F of Functional dependency if

(F1 ∪ F2 ∪ ... ∪ Fm)+ = F+.

Consider a relation R

R ---> F{...with some functional dependency(FD)..... }

R is decomposed or divided into R1 with FD { f1 } and R2 with { f2 }, thenthere can be three cases:

f1 U f2 = F -----> Decomposition is dependency preserving.

f1 U f2 is a subset of F -----> Not Dependency preserving.

f1 U f2 is a super set of F -----> This case is not possible.

Example for dependency preservation:

Dependency preservation

Example:
R=(A, B, C), F={A→B, B→C}

Decomposition of R: R1=(A, B) R2=(B, C)
Does this decomposition preserve the given dependencies?

Solution:

In R1 the following dependencies hold: F1={A→B, A→A, B→B, AB→AB}
In R2 the following dependencies hold: F2= {B→B, C→C, B→C, BC→BC}

F'= F1' ∪ F2' = {A→B, B→C, trivial dependencies}

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

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

Chapter 5

Transaction Management and Concurrency Control

Transaction Management and Concurrency Control: What is a transaction? ACID Properties, Lock Management, serializability, concurrency control with locking methods, concurrency control with time stamping methods, concurrency control with optimistic methods, specialized locking techniques.

5.1. Introduction

- A ***transaction*** is a unit of program execution that accesses and possibly updatesvarious data items.
- The transaction consists of all operations executed between the statements **begin** and **end** of the transaction
- **Transaction operations:** Access to the database is accomplished in a transactionby the following two operations:
  - ***read (X):*** *Performs the reading operation of data item X from the database*
  - ***write (X):*** *Performs the writing operation of data item X to the database*
- A transaction must see a consistent database
- During transaction execution the database may be inconsistent
- When the transaction is *committed*, the database must be consistent
- Two main issues to deal with:
  - *Failures, e.g. hardware failures and system crashes*
  - *Concurrency, for simultaneous execution of multiple transactions*

5.2 ACID Properties

- To preserve integrity of data, the database system must ensure:
  - **Atomicity:** Either all operations of the transaction are properly reflected in the database or none are
  - **Consistency:** Execution of a transaction in isolation preserves the consistency of the database
  - **Isolation:** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions; intermediate transaction results must be hidden from other concurrently executed transactions
  - **Durability:** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures
- **Example of Fund Transfer:** Let Ti be a transaction that transfers 50 from account A to B. This transaction can be illustrated as follows

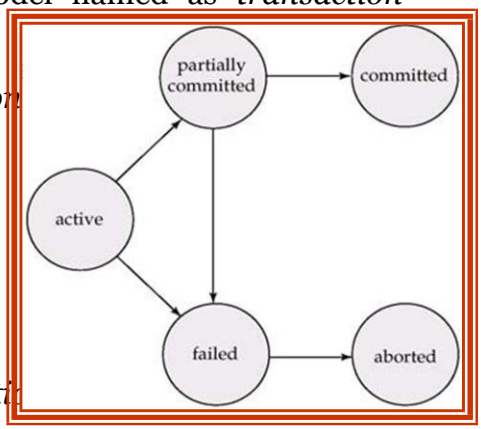
- Transfer $50 from account *A* to *B*:

  T_i : *read*(A)
  ***A* := *A* − 50**
  write(A)
  read(B)
  ***B* := *B* + 50**
  write(*B*)

- ***Consistency:*** the sum of *A* and *B* is unchanged by the execution of the transaction.
- ***Atomicity:*** if the transaction fails after step 3 and before step 6, the system shouldensure that its updates are not reflected in the database, else an inconsistency will result.
- ***Durability:*** once the user has been notified that the transaction has completed, the updates to the database by the transaction must persist despite failures.
- ***Isolation:*** between steps 3 and 6, no other transaction should access the partially updated database, or else it will see an inconsistent state (the sum *A* + *B* will be less than it should be).

Transaction and Schedules

- A transaction is seen by the DBMS as a series, or list of actions. We therefore establish a simple transaction model named as *transaction states*.



- **Transaction State:** *A transaction must be on*
- **Active,** the initial state; the transaction stays inthis state while it is executing
- **Partially committed,** after the final statementhas been executed.
- **Committed,** after *successful completion*.
- **Failed:** after the discovery that *normal executi* can no longer proceed.
- **Aborted:** after the transaction has been rolled back and the database restored to its state prior to the start of the transaction.

Concurrent Execution and Schedules

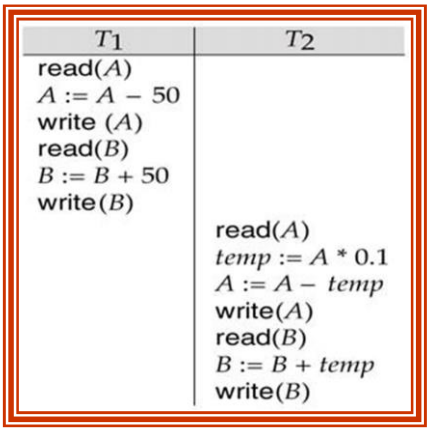
- **Concurrent execution:** executing transactions simultaneously has the followingadvantages:
  - *increased processor and disk utilization, leading to better throughput*
  - *one transaction can be using the CPU while another is reading from orwriting to the disk*
  - *reduced average response time for transactions: short transactions neednot wait behind long ones*

- **Concurrency control schemes**: these are mechanisms to achieve isolation
  - *to control the interaction among the concurrent transactions in order toprevent them from destroying the consistency of the database*

- **Schedules**: sequences that indicate the chronological order in which instructions of concurrent transactions are executed
  - *a schedule for a set of transactions must consist of all instructions of thosetransactions*
  - *must preserve the order in which the instructions appear in each individual transaction*

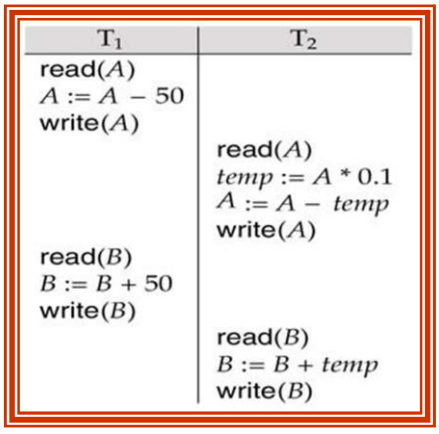
■ *Example Schedules*

- Let $T1$ transfer \$50 from A to B, and $T2$ transfer 10% of the balance from A to B. The following is a serial schedule (Schedule 1 in the text), in which $T1$ isfollowed by $T2$.



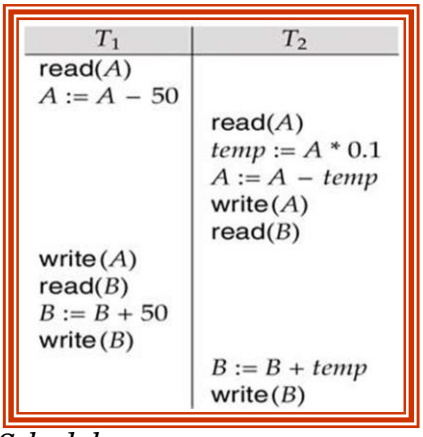
T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

- Let $T1$ and $T2$ be the transactions defined previously. The following schedule (Schedule 3 in the text) is not a serial schedule, but it is *equivalent* to Schedule 1.



T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

- The following concurrent schedule (Schedule 4 in the book) does not preserve thevalue of the the sum $A + B$



T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

■ *Serializable Schedule*

- A *serializable schedule* over a set S of committed transactions is a schedule whose effect on any consistent database is guaranteed to be identical to that of some complete serial schedule over S. i.e., even though the actions of transactions are interleaved, the result of executing transactions serially in different order may produce different results.
- **Example:** The schedule shown in the following figure is serializable.

T_1	**T_2**
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(A)
	Commit
Commit	

Even though the actions of **T_1** and **T_2** are interleaved, the result of this schedule is equivalent to first running **T_1** entirely and then running and **T_2** entirely. Actually **T_1**'s read and write of B is not influenced by **T_2**'s actions on B, and the net effect is the same if these actions are the serial schedule First **T_1**, then **T_2**. This schedule is also serializable if first **T_2**, then **T_1**. Therefore if **T_1** and **T_2** are submitted concurrently to a DBMS, either of these two schedules could be chosen as first

- A DBMS might sometimes execute transactions which is not a serial executioni.e., not serializable.
- *This can be happen for two reasons:*
  - First the DBMS might use a concurrency control method that ensures theexecuted schedule itself.
  - Second, SQL gives programmers the authority to instruct the DBMS tochoose non-serializable schedule.

■ *Anomalies due to Interleaved execution*

- There are three main situations when the actions of two transactions **T_1** and **T_2** conflict with each other in the interleaved execution on the same data object.
  - ***Write-Read (WR) Conflict:*** Reading Uncommitted data.

- ***Read-Write (RW) Conflict:*** Unrepeatable Reads
- ***Write-Write (WW) Conflict:*** Overwriting Uncommitted Data.

- *Reading Uncommitted Data (WR Conflicts)*
  - ***Dirty Read:*** The first source of anomalies is that a transaction $\mathbf{T_2}$ could read a database object A that has been just modified by another transaction $\mathbf{T_1}$, which has not yet committed, such a read is called a *dirty read.*
  - ***Example:*** Consider two transactions $\mathbf{T_1}$ and $\mathbf{T_2}$, where $\mathbf{T_1}$ stands for transferring $100 from A to B and $\mathbf{T_2}$ stands for incrementing both A and B by 6% of their accounts. Suppose that their actions are interleaved as follows:
    (i) $\mathbf{T_1}$ deducts $100 from account A, then immediately
    (ii) $\mathbf{T_2}$ reads accounts of A and B adds 6% interest to each, and then,
    (iii) $\mathbf{T_1}$ adds $100 to account B.

    This corresponding schedule is illustrated as follows:

$\mathbf{T_1}$	$\mathbf{T_2}$
R(A)	
A: = A -100	
W(A)	
	R(A)
	A: = A + 0.06 A
	W(A)
	R(B)
	B:= B+.06 B
	W(B)
R(B)	Commit
B: = B + 100	
W(B)	
Commit	

The problem here is $\mathbf{T_2}$ has added incorrect 6% interest to each A and B. Because before commitment that $100 is deducted from A, it has added 6% to account A before commitment that $100 is credited to B, it has added 6% to account

B. thus, the result of this schedule is different from the result of the other schedule which is serializable: first T1 then $\mathbf{T_2}$.

- *Unrepeatable Reads (RW Conflicts)*
  - The second source of anomalies is that a transaction $\mathbf{T_2}$ could change the value of an object A that has been read by a transaction $\mathbf{T_1}$ and $\mathbf{T_1}$ is still in progress. This situation causes a problem that, if $\mathbf{T_1}$ tries to read the value of A again, it will get a different result, even though it has not modified A in the meantime. But, this situation could not arise in a serial execute of two transactions: this, it is called as ***unrepeatable read.***

  - **Example:** Suppose that both $\mathbf{T_1}$ and $\mathbf{T_2}$ reads the same value of A, say 5. Then $\mathbf{T_1}$ has incremented A value to 6 but before commitment as A value 6, $\mathbf{T_2}$ has decremented A value from 5 to 4. Thus, instead of answer of A value as 5, i.e., from to 5 we got an answer 4 which is incorrect.

- *Overwriting Uncommitted Data (WW Conflicts)*
  - The third source of anomalies is that a transaction $\mathbf{T_2}$ could overwrite the value ofan object A, which has already been modified by a transaction $\mathbf{T_1}$, while $\mathbf{T_1}$ is still in progress.
  - **Example:** Suppose that A and B are two employees, and their salaries must be kept equal. Transaction $\mathbf{T_1}$ sets their salaries to \$1000 and transaction $\mathbf{T_2}$ sets their salaries to \$2000.

    The following interleaving of the actions T_1 and T_2 occurs:

    i) $\mathbf{T_1}$ sets A's salary to \$1000, at the same time, $\mathbf{T_2}$ sets B's salary to \$2000.

    ii) $\mathbf{T_1}$ sets B's salary is set to to \$2000, at the same time, $\mathbf{T_2}$ sets A's salaryto \$2000.

    As a result A's salary is set to \$2000 and B's salary is set to \$1000, i.e.,

    the resultis not identical

  - **Blind-Write:** Neither transaction reads a value before writing it such a write is called a ***blind-write.***
  - *The above example is the best example of blind write because T_1 and T_2 are concentrating only on writing but not on reading*.

- *Schedules involving aborted Transactions*
  - All transactions of aborted transactions are to be undone, and we can therefore imagine that they were never carried out to begin with.
  - **Example:** Suppose that transaction $\mathbf{T_1}$ deducts \$100 from account A then immediately before committing A's new value the transaction $\mathbf{T_2}$ reads the current values of accounts A and B and adds 6% interest to each, then commits, but incidentally $\mathbf{T_1}$ is aborted. So,we get incorrect result of transaction $\mathbf{T_2}$ because $\mathbf{T_1}$ was aborted in the middle of the process and $\mathbf{T_2}$ has taken incorrect value of A by $\mathbf{T_1}$ and added 6%. We say that such a schedule is ***Unrecoverable Schedule.*** The corresponding schedule is shown as follows:

$\mathbf{T_1}$	$\mathbf{T_2}$
R(A)	
A: = A -100	
W(A)	
	R(A)
	A: = A + 0.06 A
	W(A)
	R(B)
	B:= B+.06 B
	W(B)
Abort	Commit

  - Whereas, a recoverable schedule is one in which transactions read only the changes of committed transactions.

Lock-Based Concurrency Control

- Locking is a concurrent control technique used to ensure serializability.
- A lock disables occurs to data. There are two types of locks. A transaction needsto acquire a lock before performing a transaction.
- The read locks is known as shared lock and write lock is known as *exclusive lock*.
- A locking protocol is a set of rules that a transaction follows to attain Serializability.

- *Strict Two-Phase Locking (Strict 2PL) Protocol*
  - *The Strict 2PL has the following two rules:*
    - **Rule 1:** A transaction can read data only when it acquires a shared lockand can write data only when it acquires an exclusive lock on object.
    - **Rule 2:** A transaction should release the locks when it is completed.
  - The entire request for the locks is maintain by DBMS without user intervention. A transaction is blocked until it gets a requested lock.
  - If two transactions operate on two independent database objects then locking protocol allows such transactions. However if transactions operate on related data, locking protocol allows only the transaction which acquired lock.
  - **Example:** Consider two transactions, $\mathbf{T_1}$ increments the values by 10 and $\mathbf{T_2}$ increments then by 20% of the values. If the initial values of database objects A and B are 10, then after serial execution they would have 24.

$\mathbf{T_1}$	$\mathbf{T_2}$
R(A) A: = A + 10 [A = 20] W(A)	
	R(A) A: = A + 0.20 A [A = 24] W(A) R(B) B: = B + 0.20 B [B = 12] W(B) Commit
R(B) B: = B + 10 [B = 14] W(B) Commit	

Such an interleaving would yield A = 24 and B = 14 as results.

Using Strict 2PL we can avoid such anomalies. When $\mathbf{T_1}$ wishes to operate on A, it has to first acquire the key on A. When $\mathbf{T_1}$ acquires the key no other transaction can interleave.

- Using strict 2PL, the transaction first acquire a lock performs the action. However $\mathbf{T_2}$ cannot be interleaved and hence results in correct execution.

- *Deadlocks*
  - Deadlock is a situation where two or more transactions wait for locks held by other to be released.
  - **Example:** $\mathbf{T_1}$ has lock on A and $\mathbf{T_2}$ has lock on B. If $\mathbf{T_1}$ requests for lock on B by holding lock on A. Similarly, $\mathbf{T_2}$ requests for lock on B. Either $\mathbf{T_1}$ nor $\mathbf{T_2}$ can continue with the execution. This is called deadlock.
  - Deadlocks can be handled in three ways.
    i) Time-outs
    ii) Deadlock prevention
    iii) Deadlock detection and recovery

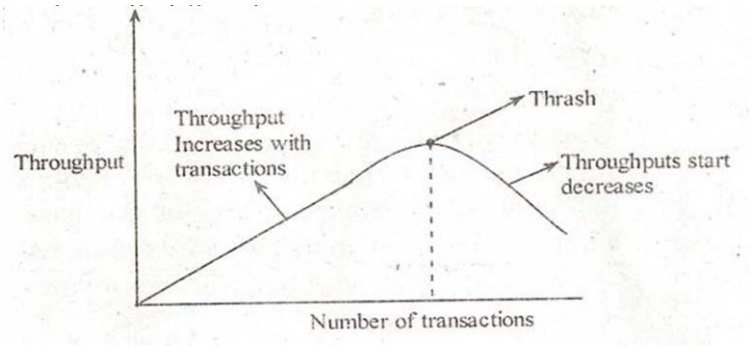
T_1	T_2
X(A) R(A) A: = A +10 W(A) X(B)	
R(B) B: = B + 10 W(B) Commit	
	X(A) R(A) A: = A + 0.20 A W(A) X(B) R(B) B:= B+.20 B W(B) Commit

- **Timeouts:** With this approach, the transaction waits for predefined amount of time before acquiring the lock. If the time-outs, DBMS assumes that so there could be a deadlock and aborts the transaction holding the object.
- **Deadlock prevention:** DBMS looks ahead to determine a deadlock. If a deadlock is predicted, then it aborts the transaction and never allows a deadlock to occur. Two algorithms are used for the purpose.
    i) Wait-Dies
    ii) Wound-Wait.
- **Deadlock Detection:** DBMS waits until a deadlock occurs and then takes measures to solve the deadlock problem. It constructs a wait- for graph (WFG) for the purpose.

Performance of Locking

- Long-based techniques use two methods to acquire serializabilty.
    i) Blocking
    ii) Aborting
    - **Blocking:** A transaction is blocked until it gets a lock for operation. Deadlock is an extreme situation where a transaction blocks forever waiting for lock. This can be avoided by aborting the transaction.
    - **Abort:** A transaction is forced to stop its execution and to restart.
- Practically, only 1% of transactions suffer from deadlocks and the transaction are aborted even less than 1%. Hence, there needs a wide consideration only on the delay introduced by blocking. These blocking delays in turn have an adverseeffect on the throughput.
- Initially the throughput of the system increases with increasing number of transactions. This is because initially transactions are unlikely to conflict. As the transactions are increased, the

throughput will not increase proportionally because of certain conflicts. As the transactions increase, there will reach a point when the system can no more handle the transactions and reduces the system throughput. This point is called *thrashes*.



- If the system reaches the thrash point, the DBA takes effective measures to reducethe number of transactions.
- The following steps are taken to increase throughput:
  i) Reducing the situation where two objects request for same lock.
  ii) Each transaction should be allowed to hold the lock for a short period oftime such that other transactions are not blocked for a long time.
  iii) Avoiding hot spots. A frequently accessed database object is known as hotspots. This hot spot reduces the system performance drastically.

5.4 Serializability

- Basic Assumption – Each transaction, on its own, preserves database consistency
  - *i.e. serial execution of transactions preserves database consistency*
- A (possibly concurrent) schedule is *serializable* if it is equivalent to a serialschedule
- Different forms of schedule equivalence give rise to the notions of *conflictserializability* and *view Serializability*.Simplifying assumptions:
  - *ignore operations other than read and write instructions*
  - *assume that transactions may perform arbitrary computations on data inlocal buffers between reads and writes*
  - *simplified schedules consist only of reads and writes*

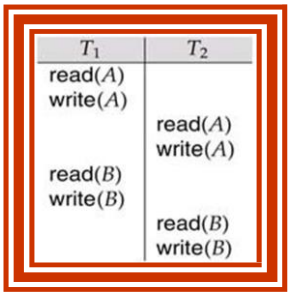
■ *Conflict Serializability*

- Instructions *li* and *lj* of transactions *Ti* and *Tj* respectively, **conflict** if and only if there exists some item *Q* accessed by both *li* and *lj*, and at least one of these instructions wrote *Q*.
  1. *li* = **read**(*Q*), *lj* = **read**(*Q*). *li* and *lj* don't conflict.
  2. *li* = **read**(*Q*), *lj* = **write**(*Q*). They conflict.
  3. *li* = **write**(*Q*), *lj* = **read**(*Q*). They conflict
  4. *li* = **write**(*Q*), *lj* = **write**(*Q*). They conflict
- Intuitively, a conflict between *li* and *lj* forces a (logical) temporal order betweenthem
- If *li* and *lj* are consecutive in a schedule and they do not conflict, their resultswould remain the same even if they had been interchanged in the ordering

- If a schedule *S* can be transformed into a schedule *S´* by a series of swaps of non-conflicting instructions, we say that *S* and *S´* are **conflict equivalent**.
- We say that a schedule *S* is **conflict serializable** if it is conflict equivalent to aserial schedule
- ***Example of a schedule that is not conflict serializable:***

T_3	T_4
read(Q)	
	write(Q)
write(Q)	

We are unable to swap instructions in the above schedule to obtain either the serial schedule < *T*3, *T*4 >,or the serial schedule < *T*4, *T*3 >.



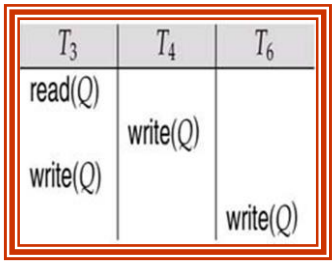
T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

- Schedule 3 below can be transformed into Schedule 1,a serial schedule where *T*2 follows *T*1, by series of swaps of non-conflicting instructions.
  Therefore Schedule 3 is conflict serializable.

■ *View Serializability*

- Let *S* and *S´* be two schedules with the same set of transactions. *S* and *S´* are *view equivalent* if the following three conditions are met, where *Q* is a data item and *Ti* is a transaction:
  1. If *Ti* reads the initial value of *Q* in schedule *S,* then *Ti* must, in schedule *S´*, also read the initial value of *Q*
  2. If *Ti* executes read(*Q)* in schedule *S*, and that value was produced by transaction *Tj* (if any), then transaction *Ti* must in schedule *S´* also read the value of *Q* that was produced by transaction *Tj*
  3. The transaction (if any) that performs the final write(*Q*) operation in schedule *S* (for any data item *Q*) must perform the final write(*Q*) operationin schedule *S´*

  NB: View equivalence is also based purely on **reads** and **writes**
- A schedule *S* is *view serializable* it is view equivalent to a serial schedule
- Every conflict serializable schedule is also view serializable
- Schedule 9 (from book) — a schedule which is view-serializable but *not* conflictserializable



T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- Every view serializable schedule that is not conflict serializable has *blind writes*

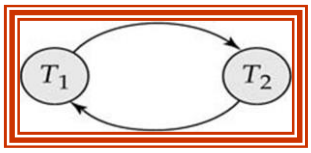
■ *Other Notions of Serializability*

- This schedule produces the same outcome as the serial schedule < $T1$, $T5$ >
- However it is not conflict equivalent or view equivalent to it
- Determining such equivalence requires analysis of operations other than read andwrite

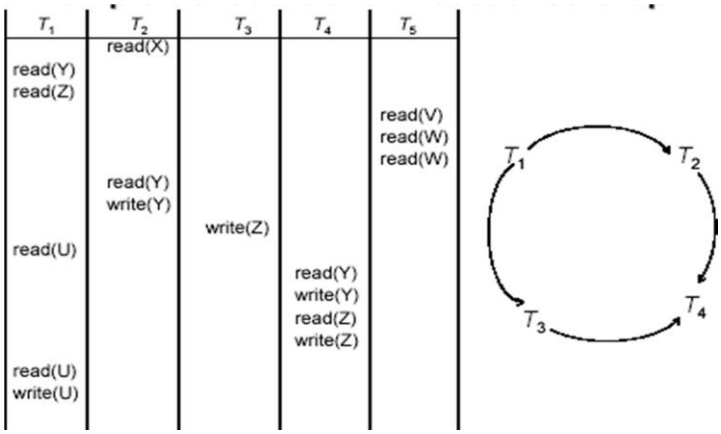
T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

■ *Testing for Serializability*

- Consider some schedule of a set of transactions $T1$, $T2$, ..., Tn
- **Precedence graph***:* a directed graph where the vertices are transaction names
- We draw an arc from *Ti* to *Tj* if the two transaction conflict, and *Ti* accessed thedata item before *Tj*
- We may label the arc by the item that was accessed
- *Example:*



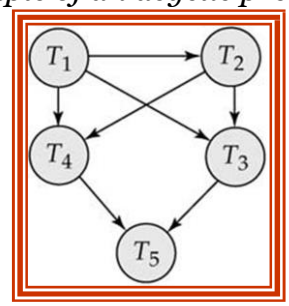
- **Example Schedule and Precedence Graph**



- A schedule is **conflict serializable** if and only if its precedence graph is acyclic
  - Cycle-detection algorithms exist which take order $n2$ time, where n is the numberof vertices in the graph
  - If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph. This is a linear order consistent with the partialorder of the graph. For example, a serializability order for this graph is $T2$ □$T1$ □ $T3$ □ $T4$ □ $T5$

- The precedence graph test for conflict serializability must be modified to apply to a test for **view serializability**
  - The problem of checking if a schedule is view serializable is *NP*-complete. Thus existence of an efficient algorithm is unlikely. However practical algorithms that just check some *sufficient conditions* for view serializability can still be used

Example of an acyclic precedence graph



- *Concurrency Control vs. Serializability Tests*
- Goal – to develop concurrency control protocols that will ensure serializability
- These protocols will impose a discipline that avoids nonseralizable schedules
- A common concurrency control protocol uses *locks*
  - *while one transaction is accessing a data item, no other transaction can modify it*
  - *require a transaction to lock the item before accessing it*
  - *two standard lock modes are "shared" (read-only) and "exclusive" (read-write)*

5.3 Lock Management

- The data items must be accessed in a mutually exclusive manner in order to ensure serializability i.e., a data item can be accessed by only one transaction at a time. This can be accomplished by allowing the transaction to access a dataitem only if it is holding a lock o that data item.
- A lock manager is a component of the DBMS that keeps track of the locks issued to the transactions. It maintains a hash tables with the data object identifier as a key called Lock Table.
- It also maintains a unique entry for each transaction in a transaction table (TT) and each entry contains a pointer to a series of locks held by the transaction.
- **A Lock Table Entry:** While can either be a page or a record for an objectcontains the following information:
  - i) The number of transactions which holds a lock on the object currentlywhich can be more than one in shared mode.
  - ii) The nature of the lock which can either be shared or exclusive and
  - iii) A pointer to a queue holding lock requests.
- *Locking Modes*

i) Shared: A transaction T-I can read the dataitem P but cannot write it,if it is holding a shared, mode lock. It is denoted by S.

ii) Exclusive: A transaction T-I can both read and write the dataitem P ifit is holding an exclusive-mode lock. It is denoted by X.

5.3.1 Implementing Lock and Unlock Requests

- According to the strict 2PL, a transaction must obtain and hold a shred (S) orexclusive (X) lock on some object 'O' before it reads or writes an object 'O'.
- A transaction T_1 can acquire the needed locks by sending a lock request to thelock manager in this manner.
    1. If a request is made for a shared lock and the request queue is empty and further the object is not locked currently in an exclusive mode then the lock manager accepts the lock requests and grant the needed lock and updates the entry for an object in the lock table.
    2. If a request is made for an exclusive lock(X) and none of the transactions is currently holding a lock on the object i.e., request queue is empty, then the lock manager grants the lock and updates the corresponding entry in the lock table.
    3. If the locks are not currently available then the request is added to the request – queue and the (requesting) corresponding transaction issuspended.
- A transaction releases all the acquired locks on its abortion or commitment. Once a lock is released a lock table entry for an object is updated by the lock manager and grants the lock to the requesting transaction present at the head of the queue.
- If more than one request is made for the shared lock then all the requests can be granted together. All the pending lock requests are queued.
- If transaction T_1 acquires a shared lock on object 'A' and if transaction T_2 requests for exclusive requests are placed in the queue, and a lock is granted when its predecessor releases the lock. Hence T2 is granted the lock when T1 unlocks.
- *Atomicity Assurance in Locking and Unlocking*
    - To ensure the atomicity of lock and unlock operations access to the lock- table can be achieved by using a semaphore which is an OS synchronization mechanism.
    - When a transaction issues an exclusive lock (X) request, the lock manager checks finds and grant the request if no other transaction is holding a lock on an object.

Lock Conversions

- If a transaction which is currently holding a shared lock on some object 'O' wants to obtain an exclusive lock on 'O' then this "upgrade lock" request is handled by granting the X on 'O' if none of the transaction holds 'S' on 'O' and no other request is present at the head of the queue. Otherwise the two transactions are deadlocked (if they request for the X on the same object).
- This deadlocks can be avoided by acquiring X locks the start-up and then downgrading them to S locks.
- **Advantage of lock-down Grading:** It improves the overall performance i.e,throughput by reducing deadlocks.

- **Disadvantage:** It reduces concurrency by acquiring write locks when they not actually needed. This drawback can be reduced by using update lock which is sent initially and prevents conflicts between the read operations else it is downgraded to a shared lock, if not needed. In case of object updates if is upgraded to X lock thereby preventing deadlocks.

Dealing with Deadlocks

- **Deadlock definition:** Deadlock is a situation where one transaction is waiting for another transaction to release locks before it can proceed.
- **Example:** Suppose a transaction T_1 holds an exclusive lock on some dataitem P and transaction T_2 holds an exclusive lock on data item Q. Now, T_1 requests an exclusive lock on Q and T_2 requests for an exclusive lock on P and are queued. So, T_1 is waiting for T-2 to release its lock and T_2 is waiting for T_1 to release its lock leading to a situation called deadlock where neither of the two transactions can proceed.
- The DBMS is responsible for the detection and prevention of such deadlocks.

5.3.2 Deadlock Prevention

- As the saying goes, prevention is better than cure, it is always better to prevent a deadlock rather than waiting for it to occur and then taking measures to avoid deadlock. It can be prevented by prioritizing the transactions.
- If a transaction T_i requests a lock which held by some other transaction T_j then the lock manager can use one of these policies.
- **Wait-Die:** Wait- Die is a non-preemptive scheme. As the name specifies, the transaction either waits for the lock or dies. The decision on whether to wait or dieis made based on the time stamp. A transaction T_2 time stamp is greater than the time stamp value of a transaction T_1 currently using the lock, then T_2 cannot wait and is rolled back.
  - **Example:** T_3 with the time stamp values as 10, 20, 30 respectively. If a transaction T_1 requests a data item which is held by T_2, then it is allowed to wait.If T_3 requests a dataitem which is held by T_2, then T_3 will be rolled back (dies).
- **Wound – Wait:** It is a preemptive scheme wherein if a transaction T_i requests a dataitem which is currently under the control of transaction T_j, it is allowed to wait if its time stamp value is greater than that of T_j, else T_j is rolled back.
  - **Example:** Consider three transactions T_1, T_2 and T_3 with the time stamp values as, respectively. If a transaction T_1 requests a dataitem which is held by T_2 then the dataitem will be preempted from T_2 and T_2 is rolled back. Also, if T_3 requests a dataitem which is held by T_2 then T_3 is allowed to wait since its time stamp is greater than that of transaction Tj.
- *Advantages of wait-die Scheme*
  i) No occurrence of deadlock, because lower priority transactions need not have towait for higher priority transactions.
  ii) It avoids starvation (i.e, no transaction is allowed to progress and rolled backrepeatedly).

- **Disadvantage:** Unnecessary rollbacks may occur
- *Advantages of Wound-wait scheme*
  1. Deadlock never occurs because higher priority transactions needn't have to wait for lower priority transaction.
  2. It also avoids starvation.
- **Disadvantage:** Unnecessary rollbacks may occur.
- **Note:** When a transaction is aborted and restarted again it should get the same time stamp as before abortion, otherwise reassignment of time stamps causes eachtransaction to become the oldest transaction.
- **Conservative 2PL:** It is a variant of 2PL that can prevent deadlock between the transactions by acquiring all the needed locks at the time of their beginning or blocking, while waiting for these locks to be available. This scheme ensures that no deadlocks can occur because a transaction acquires all the locks needed for its execution.
- **Deadlock Detection:** In order to detect and recover from the deadlocks a systemmust perform the following operations.
  1. It should maintain the information about the allocation of the data items todifferent transactions and the outstanding requests.
  2. It should provide an algorithm that determines the occurrence of deadlock.
  3. Whenever a deadlock has been detected find out the ways to recover from it. For deadlock detection the lock manager maintains a structure called a waits for graph in which nodes represents the active transactions and the are from T_i to T_j (T_i □ T_j) represents that T_k is waiting for T_j to release a lock. These edges are added to the graph by the lock manager whenever a transaction requests a lock and are removed when it grants lock requests.
- **Example:** Consider the wait – for graph as shown in figure (i)

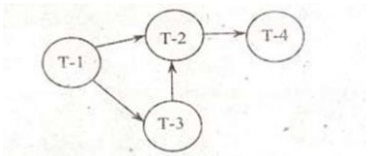


Figure: Wait for Graph with no Cycle

The following points must be noted;

i) Transaction T_1 is waiting for the transactions T_2 and T_3.
ii) Transaction T_3 is waiting for the transaction T_2.
iii) Transaction T_2 is waiting for the transaction T_4 Deadlock cannot occur asthere are no cycles is the graph.
iv) Further if a transaction T_4 is requesting for an item held by T_3 then the edge T_4□T_3 is added to the wait-for graph resulting in a new state with a cycle.

T_2□T_4□T_3□T_2 as shown below

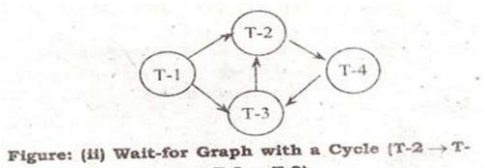


Figure: (ii) Wait-for Graph with a Cycle (T-2 → T-4 → T-3 → T-2)

The wait-for graph is checked periodically for the presence of cycles which represents deadlock. When a cycle is found, the deadlock is resolved by aborting corresponding transaction on a cycle thereby releasing its locks.

Specialized Locking Techniques

- The database objects are not always constant in the real world.
- For example, the number of customers of a bank will not be constant. Some may withdraw their accounts and some may create new accounts. Thus the number of database objects may grow of shrink. This varying number of database objects reads to a problem called Phantom problem.
- The performance of database can be enhanced by using the protocol which clearlyexplains the relationship between the objects. Two such cases include
    - Tree structured index locking
    - Collection of objects and containment relationship locking

5.3.3 The Phantom Problem and the Dynamic Database Phantom Problem

- Suppose a transaction say T1 retrieve the rows of a record that satisfy certain condition. Another transaction say T2 which is running concurrently inserts another records satisfying the same condition. If T1 retrieves the rows again, it will have a row which wasn't present previously. This differing result of same query is called phantom problem.
- Consider a student database. The principal can retrieve the records of the students at any time. The administrator creates a new record for each new student'sadmission say transaction T1 returns the details of students and transaction, T1 adds a new record to the database.
- Suppose the principal is willing to see the records of students who have scored thehighest marks. Transaction T1 acquires shared lock runs to find the student with highest score. Suppose, the highest score was found to be 500. However the administrator provides admission to a new student who secures 520 marks. And hence T2 acquires exclusive locks and updates the database.
- Now if T2 runs before T1, principal would view 520 marks. On the other hand if T1 runs before T2, principal views 500 marks.

5.3.4 Concurrency Control in B+ Trees

- **B+ Tree:** B+ tree is a balanced tree which has an equi-length for all paths from root to the leaf.
- Indices provide high level of abstraction and hence concurrency control in B+ trees ignores the index structure.
- The concurrent control in B+ trees is based on locking. The highest rend node is locked and the complete tree is traversed. Locking overhead is negligible when efficient locking protocols are used.
- Searches acquire a shared lock on the root node and proceeds further. The rood node unlocks when its child takes up the lock.
- We can also obtain an exclusive lock on all the nodes of the tree. However for insertions. Exclusive lock is required only in cases when the child node is full hence this technique is not implemented.
- The efficient technique assigns a shared lock to the root node and proceeds furtherby assigning shared lock to the child. If the child is not full, the lock on its parent is released. However if the child is full, the lock on the parent is not released.
- This is called "crabbing" or "lock-coupling"

5.3.5 Multiple Granularity Locking

- It is a technique used for locking complex objects (object within an object)
- **Example:** A University contains several colleges, each college has many courses and each course has several students. A student can select a preferred course in aparticular college.
- Similarly a database contains several files and each file contains many pages which in turn is a gap of records. This is called containment hierarchy which can be represented as a tree of objects. A transaction can obtain a lock on a selected item just as a student chooses a course of his choice locking a node in a tree involves locking all its children.
- Apart from S and X locks multiple granularity locking uses:
  (i) Intension shared (IS) and
  (ii) Intension exclusive (IX).
- Conflicts between the locks can be explained using the flowing table

	S	X
IS	Doesn't conflicts	Conflicts
IX	Conflicts	Conflicts

- Thus, if a transaction acquires on X or S lock on some node 'i', it must first lock its parents either in IS or IX.
- A transaction must obtain S and IX clock in order to read a file followed by the modification of some of its records or it can also acquire SIX lock.

SIX = S+IX

- Locks acquisition is a top-down approach whereas lock releasing is a bottom-up approach (leaf-root). Multiple granularity locking is used in conjunction with 2PL, this ensures serializability as 2PL predicts when to release the locks.
- "Granularity" of the locking is an important issue. Hence fine granularity locks are acquired in the beginning and after making some requests, the next higher level granularity locks can be obtained. This phenomenon is called lock escalation.

5.5 Concurrency control without locking

- In DBMS the concurrency can be controlled without locking also, by using the following techniques:
  - Optimistic Concurrency Control
  - Timestamp-Based Concurrency Control
  - Multiversion Concurrency Control

5.5.1 Optimistic Concurrency Control

- In this it is assumed that the conflicts between the transactions are occasionalhence there is no need for locking and time stamping.
- In this technique when a transaction reaches its COMMIT step, it is checked for the presence of conflicts. On occurrence of it the transaction must be rolled back and restarted. This happens rarely as there are very few conflicts.

- The transactions proceed in optimistic concurrency control in three phases as follows:
    - **Read:** The transaction executes, reading values from the database and writing to private workspace.
    - **Validation:** If the transaction decides that it wants to commit, the DBMS checks whether the transaction could have conflicts with any other currently executing transaction. If there is possibility to conflict, the transaction is aborted, and its private workspace is cleared and it is restarted.
    - **Write:** If validation determines that there are no possible conflicts, the changes to data objects made by the transaction in its private workspace are copied into the database.
- Remember, if there are few conflicts, then validation can be done efficiently, this approach should lead to better performance than locking. But, if there are many conflicts, the cost of repeatedly restarting transaction hurts performance.

- Thus, each transaction T_i is assigned a time stamp TS (T_i) at the beginning of its validation phase, and the validation criterion checks whether the timestamp ordering of transactions is an equivalent serial order transaction.

- For every pair of transactions T_i and T_j such that TS(T_i)< TS(T_j), one of the following validation conditions must hold:
    1. Ti completes (all three phases) before T_j begins.
    2. Ti completes before T_j starts its write phase, and Ti does not write any database objects read by T_j.
    3. Ti completes its Read phase before T_j completes its Read phase, and Ti does not write any database object, that is either read or written by T_j.
- To validate T_j, we must check to see that one of these conditions holds with respect to committed transaction T_i such that TS(T_i) < TS(T_j). Moreover, each condition ensures that T_j's modifications are not visible to T_i.
- The first condition allows T_j to see some of T_i's changes, but they execute completely in serial order with respect to each other.
- The second condition allows T_j to read objects while Ti is still modifying objects, but there is objects written by T_i, all of T_i's writes precede all of T_j's writes.
- The third condition allows T_i and T_j to write objects at the same time and thus, have even more overlap in time than the second condition, but the sets of object written by the two transactions cannot overlap.
- Thus, no RW, WR, or WW conflicts are possible if any of these three conditionsis met and the concurrency is controlled without locking through an optimistic concurrency control approach.

5.6 Time Stamp –Based Concurrency control Time Stamp

- In optimistic concurrency control, a timestamp ordering is imposed on transactions and validation checks that all conflicting actions occurred in the same order.

- So, each transaction can be assigned a timestamp at startup, and we can ensure, at execution time, that if an action a_i of transaction T_i conflicts with action a_j of transaction T_j, a_i occurs before a_j if TS(T_i)< TS(T_j). If an action violates this ordering, the transaction is aborted and restarted.

- The timestamp concurrency control is implemented by giving every database object O a read timestamp RTS(O) and a write timestamp WTS(O). If transaction T wants to read object O, and TS(T)<WTS(O), the order of the read with respect to the most recent write on O would violate the timestamp order between this transaction and the writer. Therefore, T is aborted and restarted with a new, larger timestamp, if TS(T)> WTS(O), T reads O, and RTS(O) is set to the larger of RTS(O) and TS(T).

- Now consider what happens when transaction T wants to write object O:

    1. If TS(T)<RTS(O), the write actions conflicts with the most recent read action of O, and T is therefore aborted and restarted.
    2. If TS(T)<WTS(O), a simple approach would be to abort T because it writes action conflicts with the most recent write of O and is out of timestamp order. However, we can safely ignore such writes and continue. Ignoring outdated writes is called the Thomas Write Rule.
    3. Otherwise, T writes O and WTS (O) is set to TS(T).

    - **Thomas's Write Rule:** As roll back restart doesn't occur in the time stamp method, Thomas's write rule ahs been used.

        1. When transaction i wants to write the value of some data item 'D' which is already being read by some younger transaction then it is not possible for transaction i to write its value hence, it must be aborted, rolled back and restarted with a new time stamp value.

        2. When a transaction i wants to write a new value to some data item 'D' on which the write operation has already been applied by some younger transaction then the write operation requested by the transaction i is neglected and is allowed to proceed with its normal execution.

        3. Whereas in other operations a transaction; is permitted to continue with its execution and its write time stamp is changed along with the change intransactional time stamp.

    - Thus, by using Thomas's write rule it would be possible to obtain both serializable and recoverable schedules.

- The time stamp protocol just presented above, permits schedules that are not recoverable, as illustrated in the following figure:

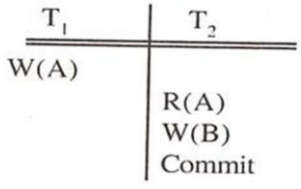


Figure : An Unrecoverable Schedule

- If TS(T_1) = and TS(T_2)=2, this schedule is permitted by the time stamp protocol (with or without the Thomas write Rule). This timestamp protocol can be modified to disallow such schedules by buffering all write actions until the transaction commits.
- In the above example, when T_1 wants to write A, WTS (A) is updated to reflect this action, but the change to A is not carried out immediately; instead, it is recorded in a private workspace, or buffer. When T2 wants to read A, then its timestamp is compared with WTS (A), and the read is seen to be permissible. However, T_2 is blocked until T_1 completes. If T_1 commits, its change to A is copies from the buffer; otherwise, the changes in the buffer are discarded. T_2 is then allowed to read A.

- This blocking of T_2 is similar to the effect of T_1 obtaining an exclusive lock on A. With this modification, the timestamp protocol permits some schedules which are not permitted by 2PL at all.
- As recoverability is essential, such a modification must be used for the timestamp protocol.

5.5.2 Multiversion concurrency control

- Multiversion concurrency control protocol represents another way of using timestamps, assigned at startup time, to achieve serializability. The goal of this protocol is to ensure that a transaction never has to wait to read a database object and the idea to maintain several versions of each database object with a write timestamp and let transaction T_i read the most recent version whose timestamp precedes TS (T_i).

- If transaction Ti wants to write an object, we must ensure that the object has not already been read by some other transaction T_j such that $TS(T_i)<TS(T_j)$.
- If we allow T_i to write such an object, its change should be seen by T_j for serializability, but T_j which read the object at some time in the past, will not see T_i's change.

- This condition can be checked, byu associating read timestamp with every object and whenever a transaction reads the object, the read timestamp is set to the maximum of the current read is aborted and restarted with a new, larger timestamp. Otherwise, Ti creates a new version of O and sets the read and write timestamps of the new version to TS(Ti).
- The drawbacks of this scheme are same, as timestamp concurrency control and in addition, there is the cost of maintaining versions. On the other hand, reads are never blocked, which can be important for workloads dominated by transaction that only read values from the database.