

# **Notes on Database Management System**

Arup Sarkar

4/6/2020

# **Notes on Database Management System**

Dr Arup Sarkar

**Important:** The following tutorial is adapted from multiple sources. All credits go to them. Students can consult these sources for better and deeper understanding of the topic. The sources are given below:

Gupta, S. B., & Mittal, A. (2017). *Introduction to Database Management System* (Second ed.). University Science Press.

Hoffer, J. A., Ramesh, V., & Topi, H. (2016). *Modern Database Management* (Twelfth, Global ed.). Pearson.

Hogan, R. (2018). *A Practical Guide to Database Design* (Second ed.). CRC Press.

ITL Education Solutions Limited. (2012). *Express Learning: Database Management Systems*. Pearson India.

Kroenke, D. M., Auer, D. J., Vandenberg, S. L., & Yoder, R. C. (2017). *Database Concepts* (Eight ed.). Pearson.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Korth* (Seventh ed.). McGraww Hill.

Tutorials Point Pvt. Ltd. (n.d.). *DBMS*. Retrieved from [www.tutorialspoint.com](http://www.tutorialspoint.com).

## **DATABASE MANAGEMENT SYSTEM**

### **Data and Information**

Historically, the term *data* referred to facts concerning objects and events that could be recorded and stored on computer media. For example, in a salesperson's database, the data would include facts such as customer name, address, and telephone number. This type of data is called *structured* data.

An expanded definition of **data** that includes structured and unstructured types is "a stored representation of objects and events that have meaning and importance in the user's environment."

The terms *data* and *information* are closely related and in fact are often used interchangeably. However, it is useful to distinguish between data and information. We define **information** as data that have been processed in such a way that the knowledge of the person who uses the data is increased.

### **What is Data Processing?**

**Data Processing** is the term generally used to describe what was done by large mainframe computers from the late 1940's until the early 1980's (and which continues to be done in most large organizations to a greater or lesser extent even today): large volumes of raw transaction data fed into programs that update a master file, with fixed-format reports written to paper.

### **What is Data Management System?**

The term **Data Management Systems** refers to an expansion of the Data Processing technique, where the raw data, previously copied manually from paper to punched cards, and later into data-entry terminals, is now fed into the system from a variety of sources, including ATMs, EFT, and direct customer entry through the Internet. The master file concept has been largely displaced by database management systems, and static reporting replaced or augmented by ad-hoc reporting and direct inquiry, including downloading of data by customers. The ubiquity of the Internet and the Personal Computer have been the driving force in the transformation of Data Processing to the more global concept of Data Management Systems.

### **What is a file-processing/file-based system? What are the limitations of a file-processing system that led to the development of the database system?**

In the file-processing or file based system, the data are stored in the form of files, and a number of application programs are written by programmers to add, modify, delete and retrieve data to and from appropriate files. New application programs are written as and when needed by the organization.

#### **Limitations of File based system:-**

1. **Data Redundancy and Inconsistency:** (Redundancy) For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost.  
(Inconsistency) For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.

2. **Difficulty in accessing data:** Conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.
3. **Data Isolation:** Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
4. **Integrity problem:** For example, the balance of a bank account may never fall below a prescribed amount (say ₹25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them.
5. **Atomicity Problems:** Consider a program to transfer ₹50 from account 'A' to account 'B'. If a system failure occurs during the execution of the program, it is possible that the ₹50 was removed from account 'A' but not credited to account 'B', resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit & debit occur, or that neither occur. That is the funds transfer must be atomic – it must happen in its entirety or not at all.
6. **Concurrent-access Anomalies:** Consider bank account A, containing ₹500. If two customers withdraw funds (say ₹50 & ₹100 respectively) from account A at about the same time, the result of the concurrent executions may leave the account in an incorrect state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value ₹500, and write back ₹450 & ₹400 respectively. Depending on which one writes the value last, the account may contain ₹450 or ₹400, rather than the correct value of ₹350.
7. **Security Problems:** In a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad-hoc manner, enforcing such security constraints is difficult.

### What is database?

A database can be defined as a collection of related data from which users can efficiently retrieve the desired information. A database can be anything from a simple collection of roll numbers, names, addresses and phone numbers of students to a complex collection of sound, images and even video or film clippings. Though databases are generally computerized, instances of non-computerized databases from everyday life can be cited in abundance. A dictionary, a phone book, a collection of recipes and a TV guide are examples of non-computerized databases. The examples of computerized databases include customer files, employee rosters, books catalog, equipment inventories and sales transactions.

Databases are organized by fields, records and files. These are described briefly as follows:

- **Fields:** It is the smallest unit of the data that has meaning to its users and is also called data item or data element. Name, Address and Telephone number are examples of fields. These are represented in the database by a value.
- **Records:** A record is a collection of logically related fields and each field is possessing a fixed number of bytes and is of fixed data type. Alternatively, we can say a record is one complete set of fields and each field have some value. The complete information about a particular phone number in the database represents a record. Records are of two types **fixed length records** and **variable length records**.

- **Files:** A file is a collection of related records. Generally, all the records in a file are of same size and record type but it is not always true. The records in a file may be of fixed length or variable length depending upon the size of the records contained in a file. The telephone directory containing records about the different telephone holders is an example of file.

### **Database Approach:**

Database is a shared collection of logically related data (and a description of this data), designed to meet the information need of an organization.

To describe the term logically we use 1. Entity, 2. Attribute, 3. Relationship

1. **Entity:** an entity is a 'thing' or 'object' in the real world that is distinguishable from other objects. For example, each person is an entity and bank accounts can be considered as entities.
2. **Attributes:** entities are described in a database by a set of attributes. For example, the attribute account-number & balance may describe one particular account in a bank, and they form attributes of the account entity set. Similarly, attributes customer-name, customer-street, address, customer-city may describe a customer entity.
3. **Relationship:** a relationship is an association among several entities. For example, a depositor relationship associates a customer with each account that s/he has.

### **What is Database Management System?**

A Database Management System (DBMS) is an integrated set of programs used to create and maintain a database. The main objective of a DBMS is to provide a convenient and effective method of defining, storing, retrieving and manipulating the data contained in the database.

### **What is DBMS catalog?**

To provide a high degree of data independence, the definition or the description of the database structure (structure of each file, the type and storage format of each data item) and various constraints on the data are stored separately in a table. This table is known as the DBMS catalog. The information contained in the catalog is called the metadata (data about data).

### **Describe Metadata.**

**Metadata** are data that describe the properties or characteristics of end-user data and the context of that data. Some of the properties that are typically described include data names, definitions, length (or size), and allowable values. Metadata describing data context include the source of the data, where the data are stored, ownership (or stewardship), and usage. Although it may seem circular, many people think of metadata as "data about data."

There are three main types of metadata:

1. **Descriptive Metadata:** It describes a resource for purpose such as discovery and identification. In a traditional library cataloging that is form of metadata, title, abstract, author and keywords are examples of meta data.
2. **Structural Metadata:** It describes how compound objects are put together. The example is how pages are ordered to form chapters.

3. **Administrative Meta data:** It provides information to help manage a resource, such as when and how it was created, file type and other technical information, and who can access it. There are several subsets of data.

#### **Instances & schemas:**

Database change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an instance of the database. The overall design of the database is called the database schema. Schemas are changed infrequently, if at all.

This concept can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an instance of a database schema.

#### **Components of the DBMS environment:**

A database system involves four major components: data, hardware, software and users.

Users can be divided into 4 groups who can use the DBMS.

1. Data & Database Administrator
2. Database Designer
3. Application Programmers
4. End-users

1. **Data Administrator's** is to decide what data should be stored in the database in the first place and to establish policies for maintaining and dealing with that data once it has been stored. An example of such a policy might be one that dictates who can perform what operations on what data in what circumstances – in other words, a data security policy.

Data administrator is a manager, not a technician. The technical person responsible for implementing the data administrator's decisions is the **database administrator (DBA)**. The DBA, unlike the data administrator, is thus an IT professional.

#### **2. Database Designer**

a) **Logical Database Designer:** is concerned with identifying the data, the relationship between the data and the constraints on the data that is to be stored in the database.

b) **Physical Database Designer:** must have a thorough and complete understanding of the organization's data and its business rules.

3. **Application Programmers** are responsible for writing database application programs in some programming language such as COBOL, C++, Java or some higher-level 'fourth generation' language. Such programs access the database by issuing the appropriate request - typically an SQL statement to the DBMS.

4. **End users** or database users are those who interact with the database in order to query and update the database and generate reports. Database users are classified into the following categories:

- **Naïve users:** The users who query and update the database by invoking some already written application programs. For example, the owner of the bookstore enters the details of various books in the database by invoking an appropriate application program.
- **Sophisticated users:** The users, such as a business analyst, scientist, etc., who are familiar with the facilities provided by a DBMS interact with the system without writing any application programs. Such users use database query language to retrieve information from the database to meet their complicated requirements.
- **Specialized users:** The users who write specialized database programs that are different from traditional data-processing applications such as banking and payroll management use simple data types. Specialized users write applications such as computer-aided design systems, knowledge-base and expert systems that store data having complex data types.

#### **Advantages of DBMS:**

The main advantage of DBMS is centralized data management where the data are stored at a centralized location and are shared among multiple users. The centralized nature of the database system provides several advantages, which overcome the limitations of the conventional file-processing system.

These advantages are as follows:

- **Controlled data redundancy:** During database design, various files are integrated, and each logical data item is stored at a central location. This eliminates replicating the data item in different files, and ensures consistency and saves the storage space.
- **Enforcing data integrity:** In the database approach, enforcing data integrity is much easier. Various integrity constraints are identified by the database designer during the database design.
- **Data sharing:** The data stored in the database can be shared among multiple users or application programs. Due to shared data, it is possible to satisfy the data requirements of the new applications without having to create any additional data or with minimal modification.
- **Ease of application development:** The application programmer develops the application programs according to the needs of the users. The other issues like concurrent access, security, data integrity, etc. are handled by the DBMS itself. This makes application development an easier task.
- **Data security:** The DBMS ensures that the only means of access to the database is through an authorized channel. To ensure security, a DBMS provides security tools such as user codes and passwords.
- **Multiple user interfaces:** DBMS provides different types of interfaces such as query languages, application program interfaces and graphical user interfaces (GUI) that include forms-style and menu-driven interfaces.
- **Backup and recovery:** The DBMS provides a backup and recovery subsystem, which is responsible for recovery from hardware and software failures.

In addition to centralized data management, DBMS also has some other advantages as follows:

- **Program-data independence:** The independence between the programs and the data is known as program-data independence. It allows changing the structure of the database without making any changes in the application programs that use the database.



- **Data abstraction:** The property of DBMS that allows program-data independence is known as data abstraction. Data abstraction allows the database system to provide an abstract view of the data to its users without giving the physical storage and implementation details.
- **Supports multiple views of the data:** A database can be accessed by many users and each of them may have a different perspective or view of the data. A database system provides a facility to define different views of the data for different users. A view is a subset of the database that contains virtual data derived from the database files but it does not exist in physical form.

### Disadvantages of DBMS

The various cost and risk factors involved in implementing a database system are:

- **High cost:** Installing a new database system may require investment in hardware and software. The DBMS requires more main memory and disk storage. Moreover, DBMS is quite expensive. Therefore, a company needs to consider the overhead cost of implementing a new database system.
- **Training new personnel:** When an organization plans to adopt a database system, it may need to recruit or hire a specialized data administration group, which can coordinate with different user groups for designing views, establishing recovery procedures and fine tuning the data structures to meet the requirements of the organization. Hiring such professionals is expensive.
- **Explicit backup and recovery:** A shared corporate database must be accurate and available at all times. Therefore, a system using on-line updating requires explicit backup and recovery procedures.
- **System failure:** When a computer system containing the database fails, all users have to wait until the system is functional again. Moreover, if DBMS or the application program fails, a permanent damage may occur to the database.

### Function of DBMS

1. Data storage, retrieval and update.
2. A user-accessible catalog.
3. Concurrency Control Service.
4. Transaction supports.
5. Recovery services.
6. Authorization services.
7. Support for data communication.
8. Integrity services.
9. Services to promote data independence.
10. Utility services.

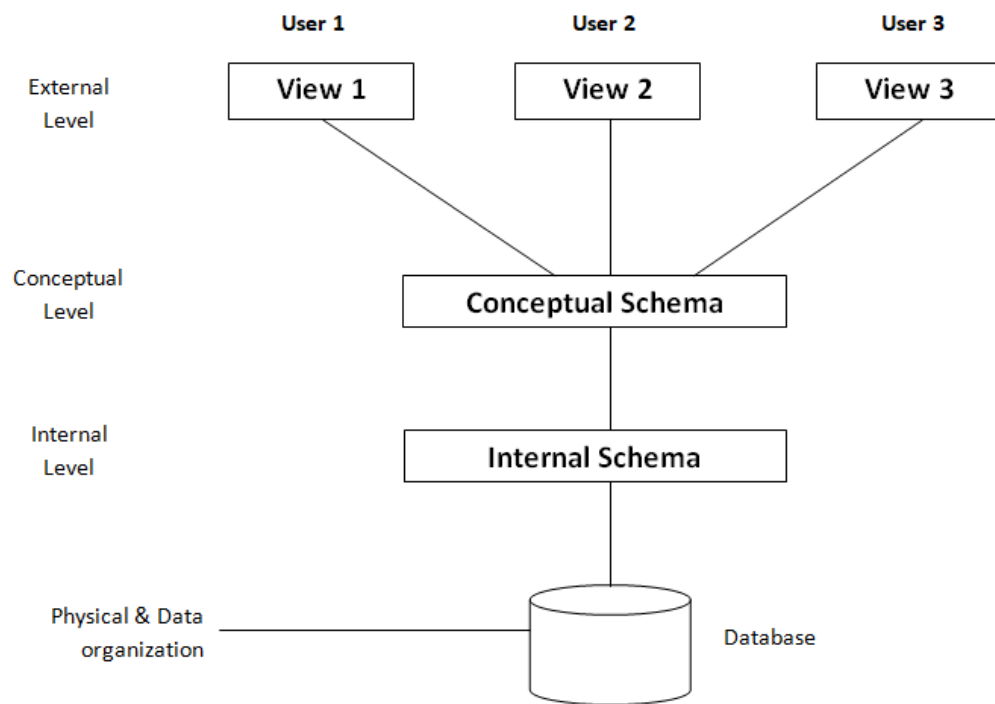
\*\*\*\*\*

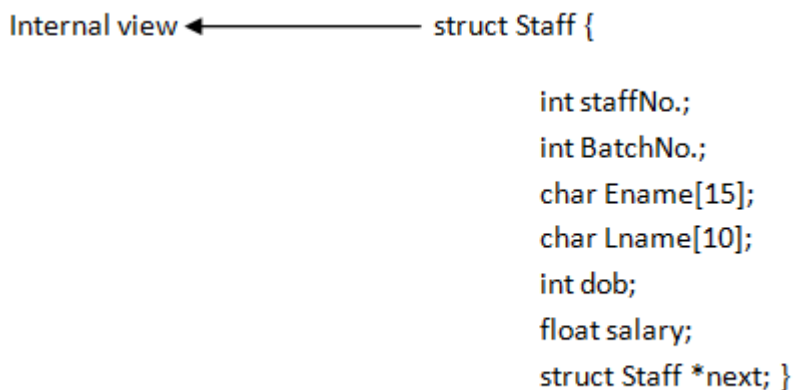
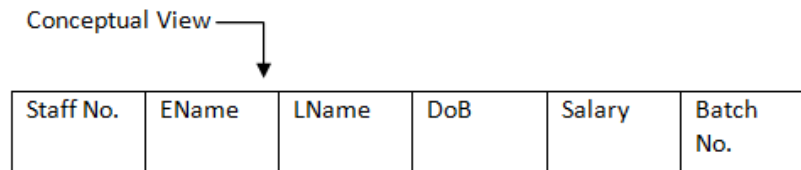
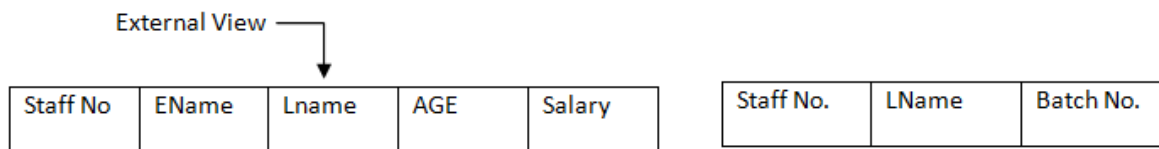
### What is the architecture of DBMS?

The architecture is divided into three levels to separate the users, application and the physical database, known as the internal, conceptual and external level.

### Why the separation is desired?

- For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify user's interactions with the system
- Each user should be able to access the same data, but in different customized view of the data. Each user should be able to change the way he or she views the data and this changes should not be affected other user.
- User should not have to deal directly with the physical database storage details such as indexing and hashing.
- The DBA should be able to change the database storage structure without affecting the user view.
- The internal architecture of the database should be unaffected by changes to physical aspects of storage, such as the change over to a new storage device.
- The DBA should be able to change the conceptual or global structure of database without affecting all users.





### Description of three level architecture:

#### **Physical level:-**

- The lowest level abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.
- Covers the physical implementation of the database to achieve optimal runtime performance and storage space utilization.
- Covers data structure and file organization used to store data on storage device.
- Interfaces with O.S. access methods to place the data on the storage devices, build the indexes, retrieve data and so on.

#### **Logic level or Conceptual level:-**

- The next higher level of abstraction describes what data are stored in the database, and what relationships exist among those data.
- This level represents all entities, attributes and their relationships.
- The constraints on the data.
- Semantic information about the data.

- Security and integrity information about the data.

#### ***View level or External level:-***

- The highest level of abstraction describes only parts of the entire database. Even though the logical level uses simple structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

#### **Mappings:**

In addition to the three levels, the architecture involves certain mappings – one conceptual to internal mapping and several external to conceptual mappings, in general

- The conceptual/internal mapping defines the correspondence between the conceptual view and the stored database. It specifies how conceptual records and fields are represented at the internal level. If the structure of the stored database is changed – i.e., if a change is made to the storage structure definition – then the conceptual/internal mapping must be changed accordingly, so that the conceptual schema can remain invariant
- An external/conceptual mapping defines the correspondence between a particular external view and the conceptual view.

#### **Data Independence:**

**Data independence** is the ability to change the schema at one level of the database system without having to change the schema at the other levels. Data independence is of two types, namely, ***logical data independence*** and ***physical data independence***.

- **Logical data independence:** It is the ability to change the conceptual schema without affecting the external schemas or application programs. The conceptual schema may be changed due to change in constraints or addition of new data items or removal of existing data items, etc. from the database. The separation of the external level from the conceptual level enables the users to make changes at the conceptual level without affecting the external level or the application programs.
- **Physical data independence:** It is the ability to change the internal schema without affecting the conceptual or external schema. An internal schema may be changed due to several reasons such as for creating additional access structure, changing the storage structure, etc. The separation of internal schema from the conceptual schema facilitates physical data independence.

*Logical data independence is more difficult to achieve than the physical data independence because the application programs are always dependent on the logical structure of the database. Data independence is an important characteristic of DBMS as it allows changing the structure of the database without making any changes in the application programs that use the database.*

In a database system, it would be extremely undesirable to allow applications to be data-dependent, for at least the following two reasons:

1. Different applications will require different views of the same data. For example, there are two applications, A and B, each owning a private file that includes the field "customer balance". Suppose, however, that application A stores that field in decimal, whereas application B stores it in Binary. It will still be possible to integrate the two files, and to eliminate the redundancy, provided the DBMS is ready and able to perform all necessary conversions between the stored representation chosen and the form in which each application wishes to see it. For example, if it is decided to store the field in decimal, then every access by B will require a conversion to or from binary.
2. The DBA must have the freedom to change the physical representation or access technique in response to changing requirements, without having to modify existing applications. For example, new kinds of data might be added to the database; new standards might be adopted; application priorities might change; new storage devices might become available; and so on.

#### **Database language:**

##### **4GL (4<sup>th</sup> Generation Language):-**

- No consensus about what constitute a 4GL.
- It is essentially a shorthand programming language.
- Compared with 3GL, which is procedural, a 4GL is non-procedural, the user defined. Explains what is to be done not how.
- Expected to rely largely on much higher level components known as fourth Generation tools.
- The user is not expected to define the steps of a program needs to perform a task but instead define parameters for the tools that used to generate an application program.
- Can improve productivity by a factor by ten.

##### **4GL encompass**

- Presentation language.
- Specialty language.
- Application generators.
- Very high level language that used to generate application code.

#### **Data Models**

Underlying the structure of a database is the data model, a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints.

Data model is consisted of three components:

- i) A structural part:- consisting of a set of rules according to which database can be constructed.
- ii) A manipulation part:- Defining the types of operations that are allowed on the data.

- iii) Possibly a set of integrity rules, which ensures that the database is accurate.

**Data models are of three categories:**

- i) Object based data model
  - ii) Record based data model
  - iii) Physical data model
- **Object based data models** use conceptual tools such as entities, attributes and relationships.
  - **Record based data models**
    - Consists of a number of fixed format records of possibly different types.
    - Each record type defines a fixed number of field, each typically of a fixed length.
    - Three principal types of record based data models:
      - Relational Data Model
      - Network Data Model
      - Hierarchical Data Model

*Relational Data Model: -*

- Based on the concept of mathematical relations.
- Entities and relationships are represented as tables each of which has a number of columns with a unique name.

**Branch**

B.No.	Street	Area	City	Pincode	TelephoneNo.	FaxNo.

**Staff**

S.No.	FName	Lname	Addr.	T.No.	Position	Sex	DoB	Salary	B.No.

*Network Data Model:-*

- Data is represented as a collection of records.
- Relationships are represented by sets.
- Records are organized as generalized graph structure where records appearing as nodes and sets as edges in the graph.

*Hierarchical Data Model:-*

- It is a restricted type of network model.
- Data is represented as collections of records.

- Relationships are represented by sets.
- Allows a node to have only one parent.
- Can be represented by a tree graph, with records appearing as nodes and sets as edges.
- **Physical Data Model**
  - Describes how data is stored in the computer representing information such as record structures, records ordering and access paths.
  - Not as many physical models as logical data model, example – unifying model and the frame memory.
- **Conceptual Modeling:**
  - Conceptual schema is the heart of the database.
  - It supports all the external view and is in turn supported by the internal schema.
  - Should be completed and accurate representation of the data requirements of the enterprise.

## Database Language

To provide various facilities to different types of users, a DBMS normally provides one or more specialized programming languages called Database (or DBMS) Languages. The DBMS mainly provides two database languages, namely, Data Definition Language (DDL) and Data Manipulation Language (DML), Data Control Language (DCL) and Transaction Control Language (TCL).

- 1) **Data Definition Language:** We specify a database schema by a set of definitions expressed by a special language called a data-definition language (DDL).

For instance, the following statement in the SQL defines the account table:

```
CREATE TABLE ACCOUNT
(account_number char(10),
Balance integer)
```

Execution of the above DDL statement creates the account table shown below.

account_number	balance
10012	5000
20023	3700
23011	2000

**The account table**

- 2) **Data Manipulation Language (DML):**

Data manipulation is

- The retrieval of information stored in the database.
- The insertion of new information into the database.
- The deletion of information from the database.

- The modification of information stored in the database.

A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. There are basically two types:

- **Procedural DMLs** require a user to specify what data are needed and how to get those data.
- **Declarative DMLs** (also referred to as nonprocedural DMLs) require a user to specify what data are needed without specifying how to get those data.

\*\*\*\*\*

### **Briefly describe conceptual data modeling.**

The overall database design and implementation process starts with requirements gathering and specifications. Once the requirements of the user have been specified, the next step is to construct an abstract or conceptual model of a database based on the requirements of the user. This phase is known as conceptual modeling or semantic modeling. Conceptual modeling is an important phase in designing a successful database. It represents various pieces of data and their relationships at a very high level of abstraction. It mainly focuses on what data are required and how they should be organized rather than what operations are to be performed on the data. The model is independent of any hardware (physical storage) and software (DBMS) constraints and hence, can be modified easily according to the changing needs of the user. The conceptual model can be represented using two major approaches, namely, Entity–Relationship Modeling and Object Modeling.

### **What is an E–R model? What are its advantages?**

The Entity–Relationship Model is the most popular conceptual model used for designing a database. It was originally proposed by Dr Peter Chen in 1976 as a way to unify the network and relational database views. The E–R model views the real world as a set of basic objects (known as entities), their characteristics (known as attributes) and associations among these objects (known as relationships). The entities, attributes and relationships are the basic constructs of an E–R model.

The E–R model has several advantages as listed below:

- It is simple and easy to understand and, thus, can be used as an effective communication tool between the database designer and the end user.
- It captures the real-world data requirements in a simple, meaningful and logical way.
- It can be easily mapped to the relational model. The basic constructs, that is, the entities and attributes of the E–R model can be easily transformed into relations (or tables) and columns (or fields) in a relational model.
- It can be used as a design plan and can be implemented in any database management software.

### **What do you mean by entity?**



An entity is a distinguishable object that has an independent existence in the real world. It includes all those 'things' of an organization about which the data are collected. For example, each book, publisher and author in an Online Book database is an entity. An entity can exist either physically or conceptually.

- **Tangible entity:** If an entity has a physical existence, it is termed as tangible or concrete entity. For example, a book, an employee, a place or a part.
- **Non-tangible:** If an entity has a conceptual existence, it is termed as non-tangible or abstract entity. For example, an event, a job title or a customer account.

### Entity type vs Entity set.

A set or a collection of entities that share the same attributes but different values is known as an entity type. For example, the set of all publishers who publish books can be defined as the entity type PUBLISHER. A specific occurrence of an entity type is called its instance. For example, the publisher Hills Publications is the instance of the entity type PUBLISHER. An entity set is a collection of all instances of a particular entity type in the database at any point of time. Each entity is referred to by its name and attribute values. An entity type describes the schema (intension) for the entity set that shares the same structure. The entity set, on the other hand, is called the extension of the entity type.

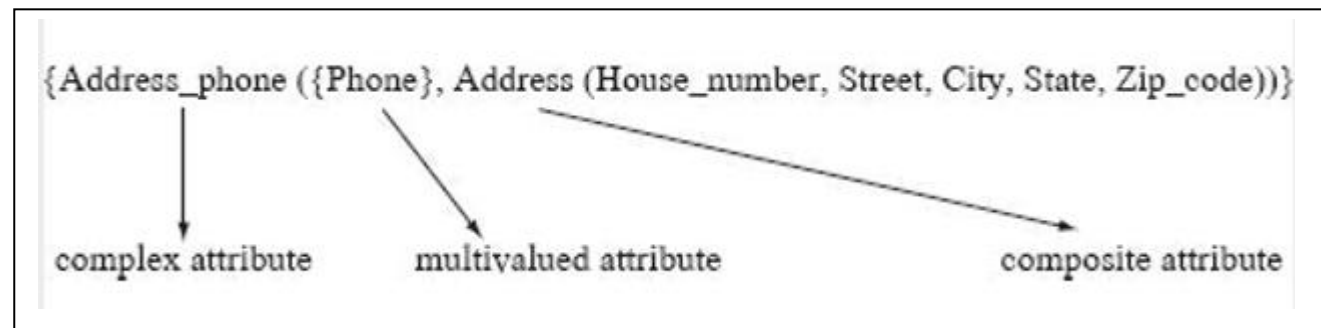
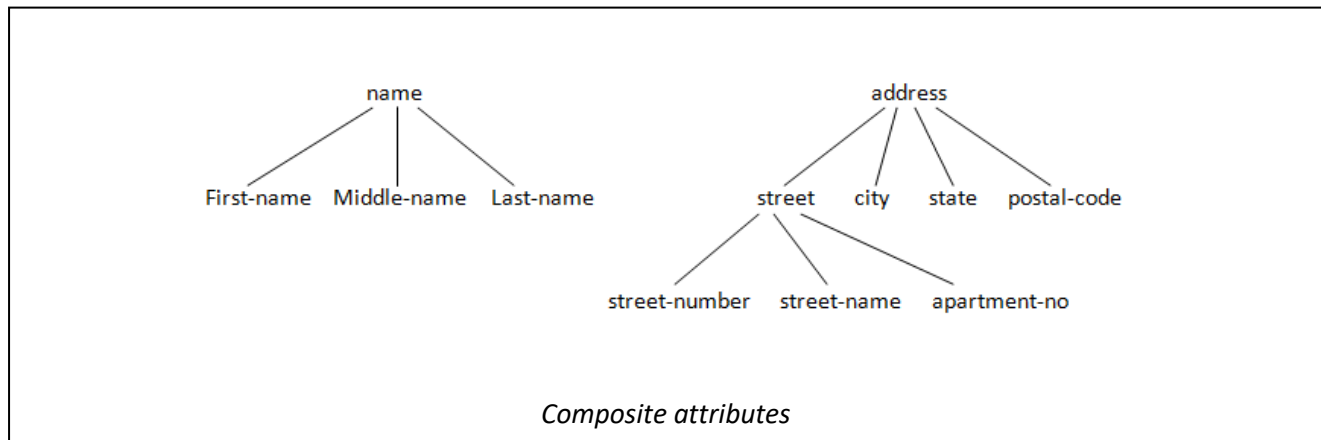
### Types of attributes.

The attributes of an entity are classified into the following categories:

- **Identifying and descriptive attributes:** The attribute that is used to uniquely identify an instance of an entity is known as an identifying attribute or simply an identifier. For example, the attribute P\_ID of the entity type PUBLISHER is an identifying attribute, as two publishers cannot have the same publisher ID. A descriptive attribute or simply a descriptor, on the other hand, describes a non-unique characteristic of an entity instance. For example, the attributes Price and Page\_count are the descriptive attributes as two books can have the same price and number of pages.
- **Simple and composite attributes:** The attributes that are indivisible are known as simple (or atomic) attributes. For example, the attributes Book\_title, Price, Year, Page\_count and Category of the entity type BOOK are simple attributes as they cannot be further divided into smaller subparts. On the other hand, composite attributes can be divided into smaller subparts. For example, the attribute Address can be further divided into House\_number, Street, City, State and Zip\_code.
- **Stored and derived attributes:** In some cases, two or more attribute values are related in such a way that the value of one attribute can be determined from the value of other attributes or related entities. Consider an entity type PERSON and its attributes Date\_of\_birth and Age. The value of the attribute Age can be determined from the current date and the value of Date\_of\_birth. Thus, the attribute Age is known as a derived attribute and the attribute Date\_of\_birth is known as a stored attribute.
- **Single-valued and multi-valued attributes:** The attributes that can have only one value for a given entity are called the single-valued attributes. For example, the attribute Book\_title is a single-valued attribute as one book can have only one title. The attributes that can have multiple values for a given entity are called multi-valued attributes. For example, the attributes

Email\_ID and Phone of the entity type PUBLISHER are multi-valued attributes, as a publisher can have zero, one or more e-mail IDs and phone numbers.

- **Complex attributes:** The attributes that are formed by arbitrarily nesting the composite and multivalued attributes are called complex attributes. This is represented by grouping the components of a composite attribute between parentheses '()' and by displaying the multi-valued attributes between braces '{}', and attributes separated by commas. For example, a publisher can have offices at different places, each with different addresses and each office having multiple phones.

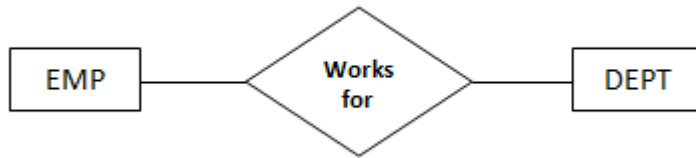


## Domain

For each attribute, there is a set of permitted values, called the domain, or value set of that attribute. The domain of attribute customer-name might be the set of all text strings of a certain length.

**Key attribute:** Key attributes are those attributes which can identify an entity uniquely in an entity set.

**Relationship:** a relationship is an association among several entities.



### Degree of relationship

The degree of a relationship is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Special cases are the binary and ternary where the degree is 2, and 3, respectively.

**Binary relationships:** the association between two entities is the most common type in the real world. A recursive binary relationship occurs when an entity is related to itself. An example might be "some employees are married to other employees".

A **ternary relationship** involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are decomposed into two or more binary relationships.

### Constraints

Relationship types usually have certain constraints that limit the possible combinations of entities participating in relationship instances.

These constraints are determined from mini-world situation that the relationship represents.

### Mapping Constraints

There are certain constraints in E-R model. Data in the database must follow the constraints. Constraints act as rules to which the contents of database must conform. There are *two types* of mapping constraints : (a) *Mapping cardinalities*, (b) *Participation constraints*.

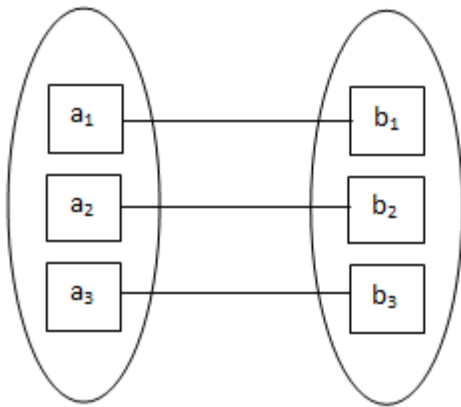
### Mapping Cardinalities and Cardinality Ratio:

Cardinality ratios express the number of entities to which another entity can be associated via a relationship set.

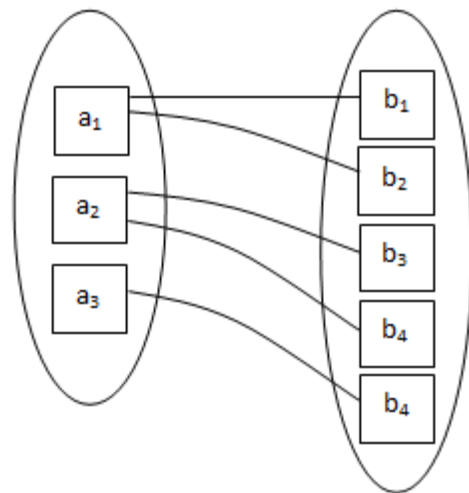
For a binary relationship set R between entity sets A and B, the mapping cardinality must be one of the following.

- i) **One to one:-** An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

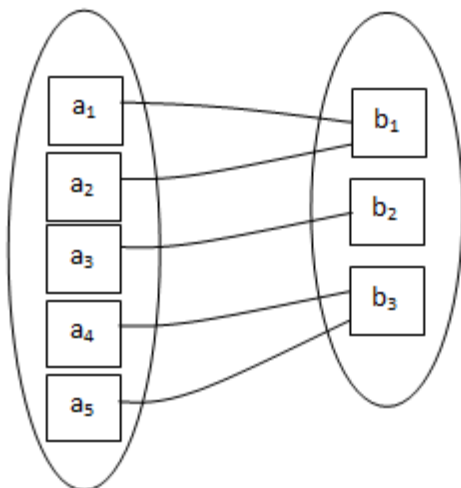
- ii) *One to many*:- an entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.
- iii) *Many to one*:- An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number of entities in A.
- iv) *Many to many*:- An entity in A is associated with any number of entities in b, and an entity in B is associated with any number of entities in A.



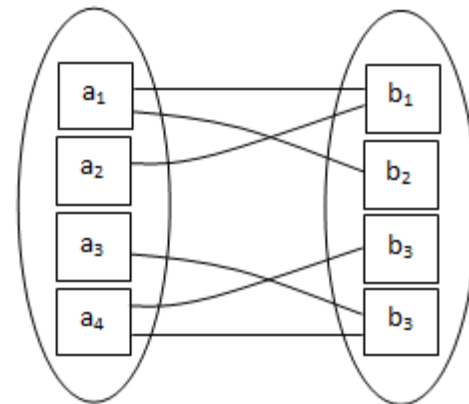
i) One to one



ii) One to many



iii) Many to one



iv) Many to many

Consider the borrower relationship set. If in a particular bank, a loan can belong to only one customer and a customer can have several loans, then the relationship set from customer to loan is one-to-many. If a loan can belong to several customers (joint venture), the relationship set is many-to-many.

**Participation constraints:**

The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R. If only some entities in E participate in relationship R is said to be partial.

For example, we expect every loan entity is related to at least one customer through the borrower relationship. Therefore the participation of loan in the relationship set borrower is total. In contrast, one individual customer of a bank may or may not take a loan. Here, only some of the customer entities are related to loan entity set through the borrower relationship, therefore the participation of customer in the borrower relationship set is partial.

### Structural constraints:

It defines as the cardinality ratio and participation constraints are taken together of a relationship type.

### Weak and strong entity set

An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed as weak entity set. An entity set that has a primary key is termed as a strong entity set.

### Keys

A key is an attribute or set of attributes that is used to identify data in entity sets. The attributes which are used as key are known as **key attributes**. Rest of all are known as **Non-key attributes**.

### Types of Keys

**Super Key:** A superkey is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.

Employee				
Reg. No	ID	Name	Salary	Dept-ID
S1D	1	Mohan	1500	10
A25	2	Sohan	2000	30
33Z	3	Vikas	3000	20
Z4X	4	Madhu	1000	10
A5C	5	Sonal	5000	20

For example, in entity set Employee, superkeys are

- (a) (ID, Name, Salary, Reg. No.)
- (b) (ID, Name, Reg. No.)
- (c) (ID) etc.

**Candidate Key:** We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called candidate keys. **Ex.** ID and Reg. No. are candidate keys.

**Primary Key:** We shall use the term primary key to denote a candidate key that is chosen by the database designer as the principal means of identifying tuples within a relation. Consider the *classroom* relation:

*classroom* (building, room-number, capacity)

Here the primary key consists of two attributes, *building* and *room-number*, which are underlined to indicate they are part of the primary key. Neither attribute by itself can uniquely identify a classroom, although together they uniquely identify a classroom.

**Foreign Key:** A foreign key is an attribute in any entity set which is also a Primary Key in any other entity set.

For example, Dept\_ID: This is an attribute in entity set Employee and also a primary key in entityset Department. Thus, it is a foreign key in Employee.

**Secondary Key:** An attribute or set of attributes which doesn't identify data uniquely but identifies a group of data is known as secondary key. For example, Name, Salary and Department No. are all secondary keys.

**Alternate Key:** All the candidate keys other than Primary Key are known as Alternate Keys. For example, If you take ID as Primary Key, then, Reg. No. is an alternate key in the Employee relation.

Department	
Dept-ID	Dept-Name
10	Sales
20	Marketing
30	Development




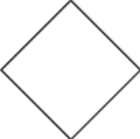




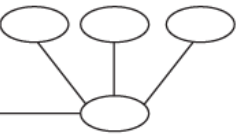



**Composite Key:** A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.

**Note:** A key (whether primary, candidate, or super) is a property of the entire relation, rather than of the individual tuples. Any two individual tuples in the relation are prohibited from having the same value on the key attributes

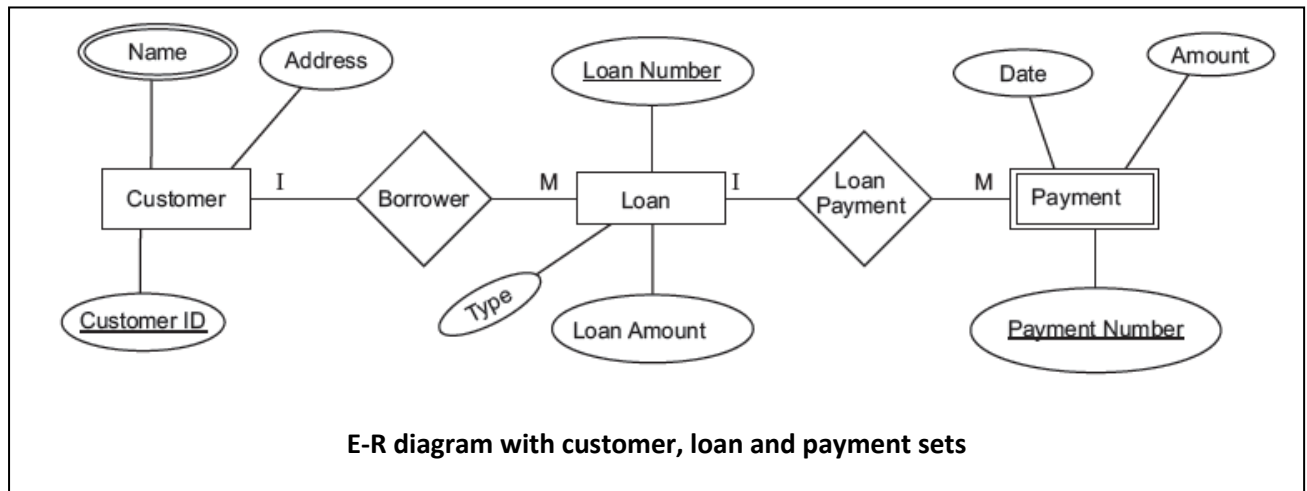
at the same time. The designation of a key represents a constraint in the real-world enterprise being modeled. Thus, primary keys are also referred to as primary key constraints.

## Entity-Relationship (ER) Model and Diagram

An **entity-relationship model (E-R model)** is a detailed, logical representation of the data for an organization or for a business area. The E-R model is expressed in terms of entities in the business environment, the relationships (or associations) among those entities, and the attributes (or properties) of both the entities and their relationships. An E-R model is normally expressed as an **entity-relationship diagram (E-R diagram, or ERD)**, which is a graphical representation of an E-R model.

S.No.	Name of Symbol	Symbol	Meaning
1.	Rectangle		Entity Set (Strong)
2.	Double Rectangle		Entity Set (Weak)
3.	Ellipse		Attribute
4.	Diamond		Relationship Set
5.	Double Diamond		Identifying Relationship Type
6.	Double Ellipses		Multi-valued attributes
7.	Dashed Ellipses		Derived attributes
8.	Ellipse with line inside it		Key attribute
9.	Ellipse joined with other ellipses		Composite attributes
10.	Double lines		Total Participation
11.	Single line		Partial Participation
12.	Triangle		Specialization or Generalization

**ER-Diagram Symbols**



### Extended E-R features

As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modeling. Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better.

Hence, as part of the **Enhanced ER Model**, along with other improvements, three new concepts were added to the existing ER Model, they were:

1. Generalization
2. Specialization
3. Aggregation

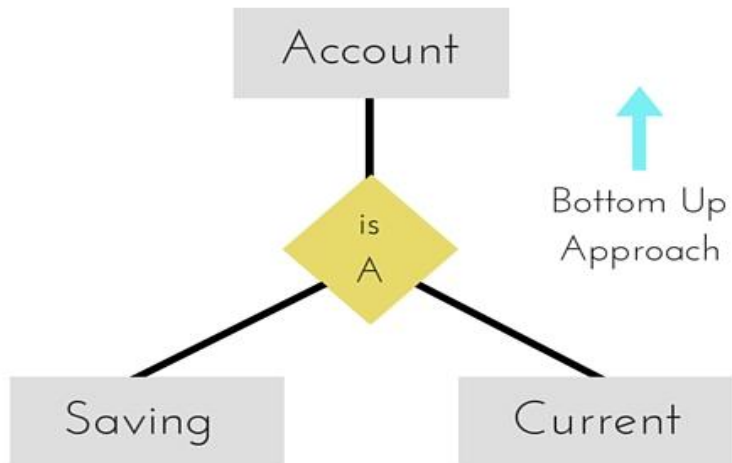
Let's understand what they are, and why were they added to the existing ER Model.

#### Generalization

**Generalization** is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entities to make further higher level entity.

It's more like Super class and Subclass system, but the only difference is the approach, which is bottom-up. Hence, entities are combined to form a more generalized entity, in other words, sub classes are combined to form a super-class. For example, **Saving** and **Current** account types entities can be generalized and an entity with name **Account** can be created, which covers both.

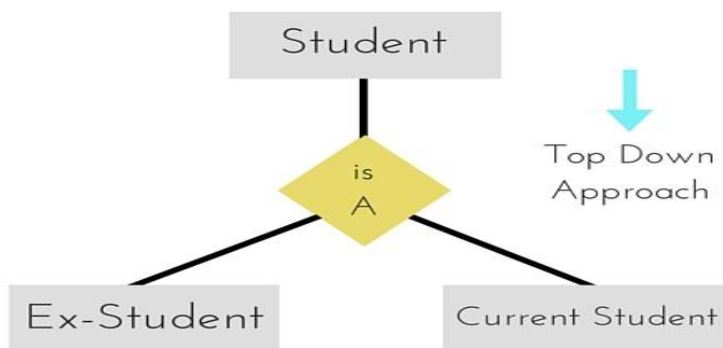




### Example of Generalization

#### Specialization

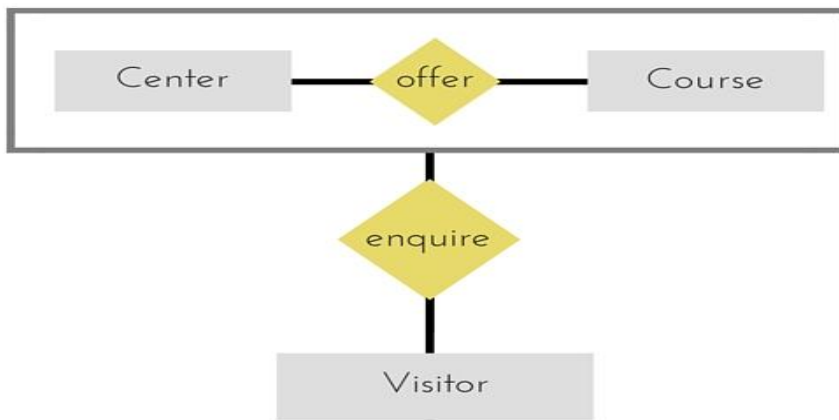
**Specialization** is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, a higher level entity may not have any lower-level entity sets, it's possible.



### Example of Specialization

#### Aggregation

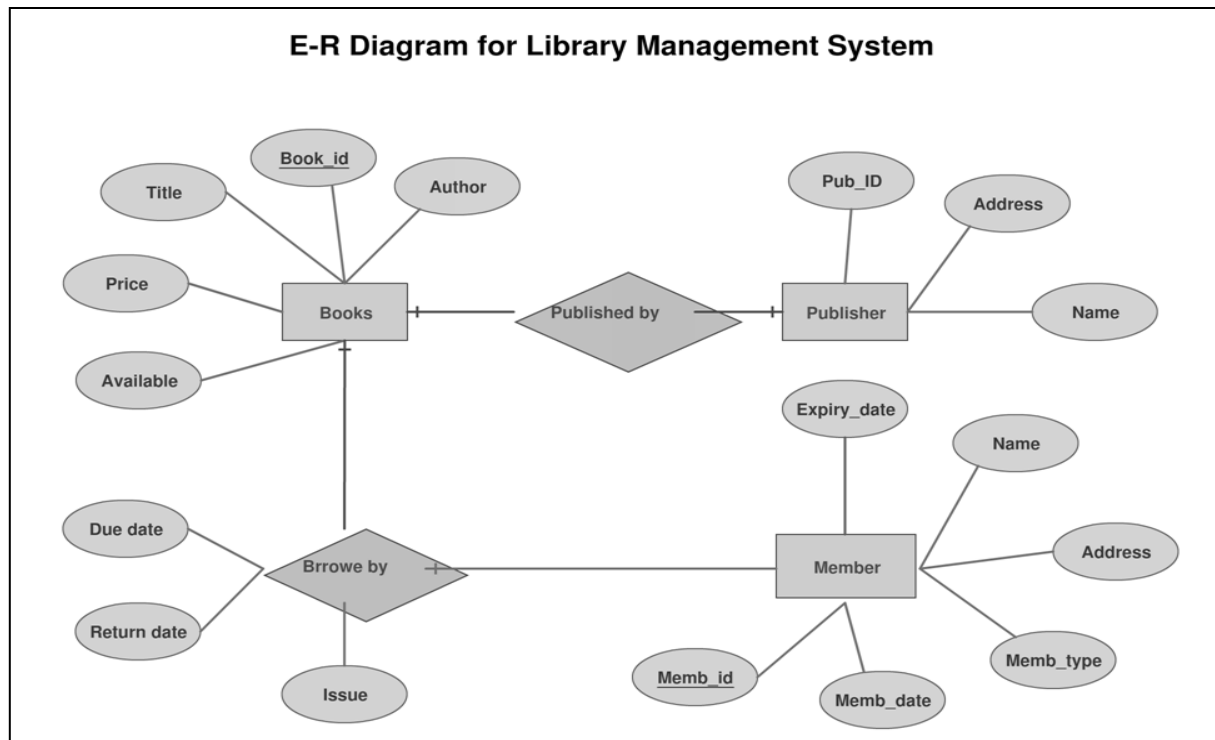
Aggregation is a process when relation between two entities is treated as a **single entity**.



### Example of Aggregation

In the diagram above, the relationship between **Center** and **Course** together, is acting as an Entity, which is in relationship with another entity **Visitor**. Now in real world, if a Visitor or a Student visits a Coaching Center, he/she will never enquire about the center only or just about the course, rather he/she will ask enquire about both.

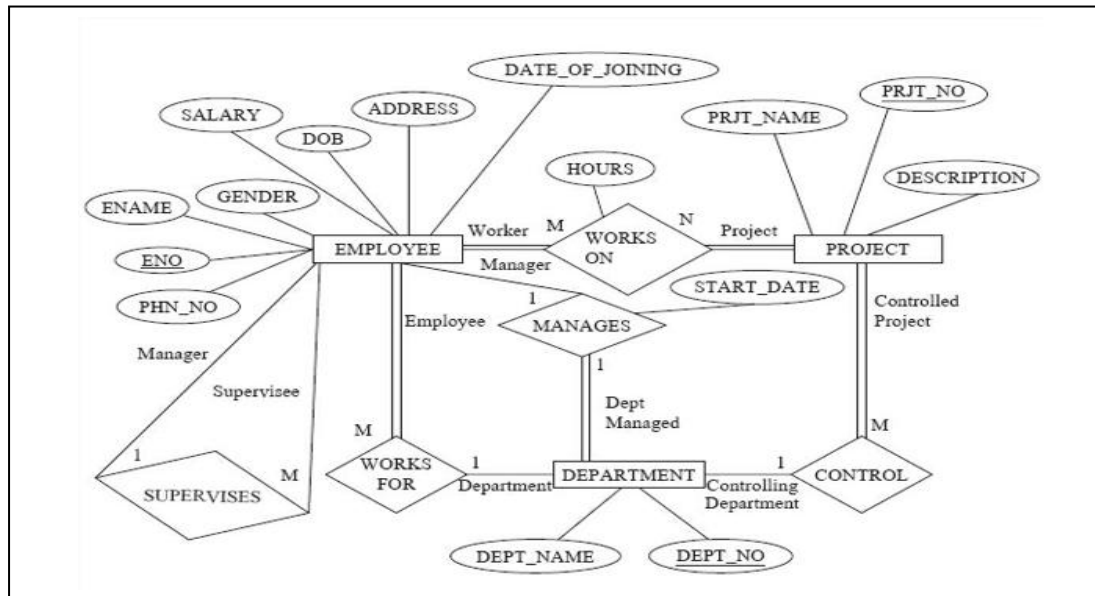
### ER Diagram of Library Management System



**Problem 1:** Design an E-R diagram for a COMPANY database as per the requirements given below. Make appropriate assumptions to complete the specification.

- The company stores the information about the currently working employees. The information includes employee number, name, gender, salary, and date of birth, date of joining, address and phone number. Each employee works for a department on a particular project for a particular number of hours.
- The information about departments includes department number and department name. Each department controls some projects currently running in the company. Also each department is managed by a particular employee who becomes the manager for that department. This employee also supervises all the other employees in that department.
- The project information includes project number, project name and its description.
- An employee can work for only one department; however, a department can have any number of employees. A department is managed by only one manager and a manager can manage only one department. A department can control any number of projects; however, one project can be handled by only one department. Any number of employees can work on any number of projects.

Specify the key attributes of each entity, role names and mapping cardinalities.



### Describe the advantages of E-R model

The major advantages of E-R model are as follows:

- Straightforward relation representation:** The relation representation of the database model using E-R diagram are relatively more straightforward than other models.
- Mapping with relational model:** It can be easily mapped onto the relational model. The E-R diagrams used in the E-R model can easily be transformed into relational tables. The entities and attributes of E-R model can easily be transformed into relations (tables) and columns (fields) in a relational model.
- Communication tool:** It is very simple and easy to understand with a minimum of training efforts required. Therefore, the model can be used by the database designer to communicate the design to the end user.
- Design tool:** E-R model can also be used as a design plan by the database developer to implement a data model in specific database management software.
- Easy conversion to other models:** E-R diagrams can be easily converted to a network or hierarchical data model.
- Graphical representation:** E-R model provides graphical and diagrammatical representation of various entities, their attributes and relationships between entities.
- Easy to modify:** Modifications to E-R diagram at later stage is relatively easier than in other models.

## Describe the limitation of E-R Model

**Limitation of E-R Model:** E-R model cannot express relationships between relationships. In other words E-R model is not capable to express relationship set between relationship sets. This limitation can be overcome by using *EER model*.

.....

## What do you understand by the term relation?

A relation is used to represent information about any entity and its relationship with other entities in the form of attributes (or columns) and tuples (or rows). It comprises a relation schema and a relation instance.

## What is a relational database?

A relational database is a collection of relations (or two-dimensional tables) having distinct names. It is a persistent storage mechanism that conforms to the relational model.

## Features of Relational database model:

- Data is presented as a collection of relations
- Each relation is depicted as a table
- Columns are attributes
- Rows ("tuples") represent entities
- Every table has a set of attributes that taken together as a "key" (technically, a "superkey") uniquely identifies each entity

## Advantages of Relational Model

The major advantages of Relational Model are as follows:

**(i) Simplicity :** The Relational database model is very easy and simple to design and implement at the logical level. The different tables in the database can be designed using appropriate attributes and data values very easily. All the relations are designed in a tabular manner, which helps the user to concentrate on the logical view of the database rather than complex internal details of how data is stored.

**(ii) Flexible :** The Relational database provide flexibility that allows changes to database structure to be easily accommodated.

**(iii) Data Independence :** Because data resides in tables, the structure of database can be changed without having to change any applications that were based on the structure. If you are using non relational database you probably have to modify the application that will access this information by including pointers to the new data. But with relational database the information is immediately accessible because it is automatically related to other data by virtue of its position in the table.

**(iv) Structural Independence :** Relational database is only concerned with data and not with the structures, which improves performance. Hence processing time and storage space is comparatively

large in relational database but the user is not required to know the details of the structure design. The structural flexibility of a relational database allows combinations of data to be retrieved that were never anticipated at the time the database was initially designed.

**(v) Query Capability** : It makes possible a high level query language *i.e.*, SQL (Structure Query Language) which avoids complex database navigation. In this model the queries are based on logical relationships and processing those queries does not require predefined access paths among the data *i.e.*, pointers.

**(vi) Matured Technology** : Relational model is useful for representing most of the real world objects and relationship between them. Relationship implementation is very easy through the use of a key.

**(vii)** Ability to easily take advantages of new hardware technology which make things easy for the users.

### **Disadvantages of Relational Model**

The major disadvantages of relational data moderate are as follows:

(i) The relational database use a simple mapping of logical tables to physical structures. Indexing and hashing techniques are used for access to table data and for certain constraint processing. This severally limits the performance.

(ii) The most significant limitation of relational model is its limited ability to deal with binary large objects such as images, spreadsheets, e-mail messages, documents etc.

(iii) Since this model has the ability to easily take advantage of new hardware technology to run smoothly, so large hardware overheads are incurred. This make it a costly affair.

(iv) Mapping objects to relational database can be a difficult skill to learn.

(v) Data Integrity is difficult to ensure with relational databases because no single application has control over the data so it is very difficult to ensure that all applications are operating under business principles. The individual database will also create problems like data duplication, data inconsistency and so on.

### **What are the Codd's Rules?**

Dr. Edgar F. CODD proposed a set of rules that are necessary for a system to qualify as a Relational Database Management System. The CODD's rules are as follows:

**Rule 0: Foundation rule.** A relational database management system must manage the database entirely through its relational capabilities.

**Rule 1: Information Rule.** The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

**Rule 2: Guaranteed Access Rule.** Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

**Rule 3: Systematic Treatment of NULL Values.** The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following: data is missing, data is not known, or data is not applicable.

**Rule 4: Active Online Catalog.** The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

**Rule 5: Comprehensive Data Sub-Language Rule.** A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

**Rule 6: View Updating Rule.** All the views of a database, which can theoretically be updated, must also be updatable by the system.

**Rule 7: High-Level Insert, Update, and Delete Rule.** A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

**Rule 8: Physical Data Independence.** The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

**Rule 9: Logical Data Independence.** The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

**Rule 10: Integrity Independence.** A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

**Rule 11: Distribution Independence.** The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

**Rule 12: Non-Subversion Rule.** If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

## DBMS vs RDBMS

S.No.	DBMS	RDBMS
1.	DBMS is a generalized software for managing and manipulating the databases.	RDBMS is a type of DBMS that depends upon the mathematical concepts of relation.
2.	DBMS's organize data by using data files with records and fields.	RDBMS's organize data by using tables, tuples and attributes.
3.	Several files cannot be stored in a single file.	Several tables can be stored in a single table known as table pools.
4.	Navigation is not so simple.	Navigation is much simpler.
5.	It is not the case here.	It creates new database tables by using basic operators.
6.	Physical and logical data independence depends upon the structure of database.	It provides physical and logical data independence by using mappings.
7.	Data handling, transaction processing and other features are less powerful than RDBMS.	RDBMS have more powerful data handling, transaction processing and other features to enhance speed, security and reliability.

## Integrity Constraints

The relational data model includes several types of constraints, or rules limiting acceptable values and actions, whose purpose is to facilitate maintaining the accuracy and integrity of data in the database. The major types of integrity constraints are domain constraints, entity integrity, and referential integrity.

- **Domain Constraints:** All of the values that appear in a column of a relation must be from the same domain. A domain is the set of values that may be assigned to an attribute. A domain definition usually consists of the following components: domain name, meaning, data type, size (or length), and allowable values or allowable range (if applicable).
- **Entity Integrity:** The entity integrity rule is designed to ensure that every relation has a primary key and that the data values for that primary key are all valid. In particular, it guarantees that every primary key attribute is non-null.
- **Referential Integrity:** In the relational data model, associations between tables are defined through the use of foreign keys. A **referential integrity constraint** is a rule that maintains consistency among the rows of two relations. The rule states that if there is a foreign key in one relation, either each foreign key value must match a primary key value in another relation or the foreign key value must be null.

## The Relational Algebra

The relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result. Some of these operations, such as the select, project, and rename operations, are called *unary* operations because they operate on one relation. The other operations, such as union, Cartesian product, and set difference, operate on pairs of relations and are, therefore, called *binary* operations.

Although the relational algebra operations form the basis for the widely used SQL query language, database systems do not allow users to write queries in relational algebra.

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

**The *instructor* relation.**

## The Select Operation

The select operation selects tuples that satisfy a given predicate. We use the lowercase Greek letter sigma ( $\sigma$ ) to denote selection. The predicate appears as a subscript to  $\sigma$ . The argument relation is in parentheses after the  $\sigma$ . Predicate is propositional 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 *instructor* relation where the instructor is in the “Physics” department, we write:

$$\sigma_{dept\_name = "Physics"}(instructor)$$

Output would be as follows:

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
33456	Gold	Physics	87000



To find the instructors in Physics with a salary greater than ₹90,000, we write:

$$\sigma_{dept\_name = "Physics" \wedge salary > 90000}(instructor)$$

### The Project Operation

Suppose we want to list all instructors' *ID*, *name*, and *salary*, but we do not care about the *dept name*. The project operation allows us to produce this relation. The project operation is a unary operation that returns its argument relation, with certain attributes left out. Since a relation is a set, any duplicate rows are eliminated. Projection is denoted by the uppercase Greek letter pi ( $\Pi$ ).

We write the query to produce such a list as:

$$\Pi_{ID, name, salary}(instructor)$$

Output

<i>ID</i>	<i>name</i>	<i>salary</i>
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

### Composition of Relational Operations

The fact that the result of a relational operation is itself a relation is important. Consider the more complicated query “Find the names of all instructors in the Physics department.” We write:

$$\Pi_{name}(\sigma_{dept\ name = "Physics"}(instructor))$$

In general, since the result of a relational-algebra operation is of the same type (relation) as its inputs, relational-algebra operations can be composed together into a relational-algebra expression.

### The Cartesian-Product Operation

The Cartesian-product operation, denoted by a cross ( $\times$ ), allows us to combine information from any two relations. We write the Cartesian product of relations  $r_1$  and  $r_2$  as  $r_1 \times r_2$ .

For example, following expression Yields a relation, which shows all the books and articles written by Rabindranath.

$$\Pi_{\text{author}} = \text{'Rabindranath'}(\text{Books X Articles})$$

## The Join Operation

The *join* operation allows us to combine a selection and a Cartesian product into a single operation. Suppose we want to find the information about all instructors together with the *course\_id* of all courses they have taught. We need the information in both the *instructor* relation and the *teaches* relation to compute the required result.

Thus, it can equivalently be written as,

$$\text{instructor} \bowtie \text{instructor.ID} = \text{teaches.ID teaches}$$

## Set Operations

Consider the following relations

Employee			Student		
EID	Name	Salary	SID	Name	Fees
1E	John	10,000	1S	Smith	1,000
2E	Ramesh	5,000	2S	Vijay	950
3E	Smith	8,000	3S	Gaurav	2,000
4E	Jack	6,000	4S	Nile	1,500
5E	Nile	15,000	5S	John	950

**The union operation :** The union operation is a binary operation that is used to find union of relations. Here relations are considered as sets. So, duplicate values are eliminated. It is denoted by ( $\cup$ ).

**Conditions for union operation :** There are two necessary conditions for union operation.

- (i) Both the relations have same number of attributes.
- (ii) Data types of their corresponding attributes must be same.

Two relations are said to be union compatible if they follow the above two conditions.

**Ex.** If you want to find the names of all employees and names of all students together then the query is

$$\pi_{\text{Name}}(\text{Employee}) \cup \pi_{\text{Name}}(\text{Student})$$

Output of the query is

Name
John
Ramesh
Smith
Jack
Nile
Vijay
Gaurav

**Set intersection operation** : Set intersection is used to find common tuples between two relations. It is denoted by ( $\cap$ ). If you want to find all the employees from Relation Employee those are also students. Rules of set union operations are also applicable here. Then the query, is  $\pi_{Name} (Employee) \cap \pi_{Name} (Student)$

$$\pi_{Name} (Employee) \cap \pi_{Name} (Student)$$

Output of the query is

Name
John
Smith
Nile

**Set-difference operation** : Set-difference operation is a binary operation which is used to find tuples that are present in one relation but not in other relation. It is denoted by ( $-$ ). It removes the common tuples of two relations and produce a new relation having rest of the tuples of first relation.

**Ex.** If you want the names of those employees that are not students, then the query, is  $\pi_{Name} (Employee) - \pi_{Name} (Student)$

$$\pi_{Name} (Employee) - \pi_{Name} (Student)$$

Output of the query is

Name
Ramesh
Jack

## Rename Operation

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter  **$\rho$** .

Example:  $\rho_x(E)$  returns the relational algebra expression E under the name x. If a relational algebra expression E (which is a relation) has the arity k, then

$$\rho_x(A_1, A_2, \dots, A_k)(E)$$

returns the expression E under the name x, and with the attribute names  $A_1, A_2, \dots, A_k$ .

## Outer join

Outer Join is an extension of natural join operations. It deals with the missing information caused by natural join operation. There are **three** types of outer joins.

**Left outer join** : It is used to take all tuples of relation that are on the left side whether they are matching with tuples of right side relation or not. It is denoted by ( $\bowtie$ ).

(Employee  $\bowtie$  Student) gives

EID	SID	Name	Salary	Fees
1E	5S	John	10,000	950
2E	NULL	Ramesh	5,000	NULL
3E	1S	Smith	8,000	1,000
4E	NULL	Jack	6,000	NULL
5E	4S	Nile	15,000	1,500

**Right outer join** : It is used to take all tuples of relation that are on the right side whether they are matching with tuples of left side relation or not. It is denoted by ( $\bowtie$ ).

(Employee  $\bowtie$  Student) gives

EID	SID	Name	Salary	Fees
3E	1S	Smith	8,000	1,000
NULL	2S	Vijay	NULL	950
NULL	3S	Gaurav	NULL	2,000
5E	4S	Nile	15,000	1,500
1E	5S	John	10,000	950

**Full outer join** : It is used to take all tuples from left and right relation whether they match with each other or did not match. It is denoted by ( $\bowtie$ ).

(Employee  $\bowtie$  Student) gives

EID	SID	Name	Salary	Fees
1E	5S	John	10,000	950
2E	NULL	Ramesh	5,000	NULL
3E	1S	Smith	8,000	1,000
4E	NULL	Jack	6,000	NULL
5E	4S	Nile	15,000	1,500
NULL	2S	Vijay	NULL	1,000
NULL	3S	Gaurav	NULL	2,000

### The Assignment Operation

It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables. The assignment operation, denoted by  $\leftarrow$ , works like assignment in a programming language.

.....

### Decomposition

Let  $U$  be a relation schema. A set of relation schemas  $\{R_1, R_2, \dots, R_n\}$  is a decomposition of  $U$  if and only if  $U = R_1 \cup R_2 \cup \dots \cup R_n$

### Lossless-Join Decomposition

A decomposition  $\{R, T\}$  of  $U$  is a lossless-join decomposition (with respect to a set of constraints) if the constraints imply that  $u = r \bowtie t$  for all possible instances of  $R, T$ , and  $U$ .

The decomposition is said to be lossy otherwise.

It is always the case for any decomposition  $\{R, T\}$  of  $U$  that  $u \subseteq r \bowtie t$ .

### Prime and Non-Prime Attributes

For a given relation  $R = \{A_1, A_2, A_3, \dots, A_n\}$ , an attribute  $A$  is a prime attribute if  $A$  is a part of any candidate key of  $R$  otherwise  $A$  is a non-prime attribute.

### Normalization

- Normalization is a process to eliminate data redundancy to get rid of three anomalies:
  - Insert anomaly
  - Update anomaly

- Delete anomaly
- Normalization basically decomposes an inconsistent relation into multiple ones.
- Benefits: data integrity, scalability and efficient storage of data.

Normalization is the process of modifying a relation schema based on its FDs and primary keys so that it conforms to certain rules called normal forms. It is conducted by evaluating a relation schema to check whether it satisfies particular rules and if not, then decomposing the schema into a set of smaller relation schemas that do not violate those rules. Normalization can be considered as fundamental to the modelling and design of a relational database, the main purpose of which is to eliminate data redundancy and avoid data update anomalies. A relation is said to be in a particular normal form if it satisfies certain specified constraints. Each of the normal forms is stricter than its predecessors. The normal forms are used to ensure that various types of anomalies and inconsistencies are removed from the database.

### **Normal Forms**

- There are basically five normal forms exists.
- Edgar F. Codd introduced the first three normal forms i.e. 1NF, 2NF and 3NF in 1972.
- In 1974 Boyce & E.F. Codd introduced BCNF(Boyce-Codd Normal Form)
- A stronger version of 3NF
- Beside this we also have 4NF, 5NF and DKNF(Domain Key Normalization Form)
- In practical 3NF is considered to achieve the sufficient amount of consistency.

### **What are the benefits of normalization?**

The benefits of normalisation include

- (a) Normalisation produces smaller tables with smaller rows, this means more rows per page and hence less logical I/O.
- (b) Searching, sorting, and creating indexes are faster, since tables are narrower, and more rows fit on a data page.
- (c) The normalisation produces more tables by splitting the original tables. Thus there can be more clustered indexes and hence there is more flexibility in tuning the queries.
- (d) Index searching is generally faster as indexes tend to be narrower and shorter.
- (e) The more tables allow better use of segments to control physical placement of data.
- (f) There are fewer indexes per table and hence data modification commands are faster.
- (g) There are small number of null values and less redundant data. This makes the database more compact.
- (h) Data modification anomalies are reduced.
- (i) Normalization is conceptually cleaner and easier to maintain and change as the needs change.

### What is functional dependency? Give examples.

- It basically describes relationship between attributes.
- Suppose in a relation R, X and Y are the non-empty set of attributes. T1 and T2 are any two tuples where  $T1.X = T2.X$  and  $T1.Y = T2.Y$ , then  $X \rightarrow Y$  i.e.
  - X determines Y or
  - Y is functionally determined by X
- Left side of " $\rightarrow$ " is known as determinant.

ISBN	Price	Page_count	R_ID	City	Rating
001-987-760-9	25	800	A002	Atlanta	6
001-987-760-9	25	800	A008	Detroit	7
001-354-921-1	22	200	A006	Albany	7
002-678-980-4	35	860	A003	Los Angeles	5
002-678-980-4	35	860	A001	New York	2
002-678-980-4	35	860	A005	Juneau	7
004-765-409-5	26	550	A003	Los Angeles	4
004-765-359-3	40	650	A007	Austin	3
003-456-433-6	30	500	A010	Virginia Beach	5
001-987-650-5	35	450	A009	Seattle	8
002-678-880-2	25	400	A006	Albany	4
003-456-533-8	30	500	A004	Seattle	9

**BOOK\_REVIEW\_DETAIL**

Consider the relation schema **BOOK\_REVIEW\_DETAIL**(ISBN, Price, Page\_count, R\_ID, City, Rating) that represents review details of the books by the reviewers. The primary key of this relation is the combination of ISBN and R\_ID. Note that the same book can be reviewed by different reviewers and the same reviewer can review different books. An instance of this relation schema is shown in Figure 6.3. In this relation, the attribute Price is functionally dependent on the attribute ISBN ( $ISBN \rightarrow Price$ ), as tuples having the same value for the ISBN have the same value for the Price. For example, the tuples having ISBN 001-987-760-9 have the same value 25 for Price. Similarly, the tuples having ISBN 002-678-980-4 have the same value 35 for Price. Other FDs such as  $ISBN \rightarrow Page\_count$ ,  $Page\_count \rightarrow Price$ ,  $R\_ID \rightarrow City$  are also satisfied. In addition, one more functional dependency  $\{ISBN, R\_ID\} \rightarrow Rating$  is also satisfied. That is, the attribute Rating is functionally dependent on the composite attribute {ISBN, R\_ID}.

### Trivial and Non-Trivial FD

- If  $X \rightarrow Y$  and  $Y$  is subset of  $X$  then  $X \rightarrow Y$  is trivial FD.
- Example:  $ab \rightarrow b$
- If  $X \rightarrow Y$  and  $X \cap Y = \emptyset$  then it is called non-trivial FD.

### Define the closure of a set of FDs.

For a given set  $F$  of functional dependencies, some other dependencies also hold. That is, some functional dependencies are implied by  $F$ . For example, consider a relation schema  $R(A, B, C)$ , such that both the FDs  $A \rightarrow B$  and  $B \rightarrow C$  hold for  $R$ . Then, FD  $A \rightarrow C$  also holds on  $R$  as  $C$  is dependent on  $A$  transitively, via  $B$ . Thus, it can be said that FD  $A \rightarrow C$  is implied. The set of all FDs that are implied by a given set  $F$  of FDs is called the closure of  $F$ , denoted as  $F^+$ . For a given  $F$  (if it is small),  $F^+$  can be computed directly. However, if  $F$  is large, then some inference rules for functional dependencies are used to compute  $F^+$ .

### Armstrong's Axioms

The following three rules called **inference axioms** or **Armstrong's Axioms** can be used to find all the FDs logically implied by a set of FDs. Let  $X, Y, Z$ , and  $W$  be subsets of attributes of a relation  $R$ . The following axioms hold:

1. **Reflexivity.** If  $Y$  is a subset of  $X$ , then  $X \rightarrow Y$ . This also implies that  $X \rightarrow X$  always holds. Functional dependencies of this type are called **trivial functional dependencies**.
2. **Augmentation.** If  $X \rightarrow Y$  holds and  $Z$  is a set of attributes, then  $ZX \rightarrow ZY$ .
3. **Transitivity.** If  $X \rightarrow Y$  holds and  $Y \rightarrow Z$  holds, then  $X \rightarrow Z$  holds.

These rules are **Sound and Complete**. They are sound because they do not generate any invalid functional dependencies. They are complete because they allow us to generate  $F^+$  (closure of  $F$ ) from the given set of functional dependencies  $F$ . It is very cumbersome and complex to use the **Armstrong's Axioms** directly for the computation of  $F^+$ . So some more axioms are added to simplify the process of computation of  $F^+$ . These additional axioms can be proved correct by using the **Armstrong's Axioms**. The additional axioms are

4. **Additivity or Union.** If  $X \rightarrow Y$  and  $X \rightarrow Z$ , then  $X \rightarrow YZ$  holds.
5. **Projectivity or Decomposition.** If  $X \rightarrow YZ$  holds, then  $X \rightarrow Y$  and  $X \rightarrow Z$  also holds
6. **Pseudotransitivity.** If  $X \rightarrow Y$  and  $ZY \rightarrow W$  holds, then  $XZ \rightarrow W$  holds.

These additional axioms are also **Sound and Complete**.

**Example.** Let  $R = (A, B, C, D)$  and  $F$  be the set of functional dependencies for  $R$  given by  $\{A \rightarrow B, A \rightarrow C, BC \rightarrow D\}$ . Prove  $A \rightarrow D$ .

**Sol.** Given set of functional dependencies for a relation  $R$  is  $\{A \rightarrow B, A \rightarrow C, BC \rightarrow D\}$

By using Armstrong Axioms, named projectivity, we can show that  $A \rightarrow BC$  (as  $A \rightarrow B, A \rightarrow C$ )



Since  $BC \rightarrow D$ , so by transitivity rule,  $A \rightarrow BC$  and  $BC \rightarrow D$  means  $A \rightarrow D$ . Hence proved.

### Covers

Consider two sets of FD's  $F_1$  and  $F_2$  over a relation scheme  $R$ . The two sets  $F_1$  and  $F_2$  are equivalent if the closure of  $F_1$  is equal to the closure of  $F_2$  i.e.  $F_1^+ = F_2^+$ . The set  $F_1$  covers

$F_2$  and  $F_2$  covers  $F_1$  iff  $F_1$  and  $F_2$  are equivalent.

**Importance of Cover:** Sometimes closure of a set of FD's  $F^+$  can be very large and difficult to compute. In that case the cover is used. It acts as a representative of the closure of  $F$ .

**Example.** Consider two sets of FDs,  $F$  and  $G$ ,  $F = \{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$  and  $G = \{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$

Are  $F$  and  $G$  equivalent?

**Sol.** To determine their equivalence, we need to prove that  $F^+ = G^+$ . However, since computing  $F^+$  or  $G^+$  is computationally expensive, there is another method. Two sets of FDs,  $F$  and  $G$  are equivalent, if all FDs in  $F$  can be inferred from the set of FDs in  $G$  and vice versa.

To see if all FDs in  $F$  are inferred by  $G$ , compute attribute closure for attributes on the

LHS of FDs in  $F$  using FDs in  $G$  (Checking whether  $F$  covers  $G$ ):

$A^+$  using  $G = ABCD$ ;  $A \rightarrow A$ ;  $A \rightarrow B$ ;  $A \rightarrow C$ ;  $A \rightarrow D$ ;

$B^+$  using  $G = BC$ ;  $B \rightarrow B$ ;  $B \rightarrow C$ ;

$AC^+$  using  $G = ABCD$ ;  $AC \rightarrow A$ ;  $AC \rightarrow B$ ;  $AC \rightarrow C$ ;  $AC \rightarrow D$ ;

Thus, all FDs in  $F$  can be inferred using FDs in  $G$ .

To see if all FDs in  $G$  are inferred by  $F$ , compute attribute closure for attributes on the LHS of FDs in  $G$  using FDs in  $F$  (Checking whether  $G$  covers  $F$ ):

$A^+$  using  $F = ABCD$ ;  $A \rightarrow A$ ;  $A \rightarrow B$ ;  $A \rightarrow C$ ;  $A \rightarrow D$ ;

$B^+$  using  $F = BC$ ;  $B \rightarrow B$ ;  $B \rightarrow C$ ;

Since all FDs in  $F$  can be obtained from  $G$  and vice versa, hence  $F$  and  $G$  are equivalent.

### First Normal Form (1NF)

A relation is said to be in 1NF if,

- Every attribute value is atomic (single).
- There is no repeating groups exists.
- Each row is uniquely identifiable.
- Attribute names are unique as well.

The Student\_details relation (shown in Figure (a)) contains non-atomic values; hence, it is unnormalized. However, it can be normalized into 1NF by creating one tuple for each value in multi-valued attributes as shown in Figure b.

**Student\_details:**

Std_id	Std_name	Course_id	Course_title
1	A	C01	History
		C02	Geography
		C03	English
2	B	C01	History
		C04	Economics
		C05	Chemistry

(a) Unnormalised since Course\_id is not atomic

**Student\_details:**

Std_id	Std_name	Course_id	Course_title
1	A	C01	History
1	A	C02	Geography
1	A	C03	English
2	B	C01	History
3	B	C04	Economics
4	B	C05	Chemistry

(b) Normalised version of Student\_details

**Partial Dependency:**

When there is a functional dependence in which the determinant is only part of the primary key, then there is a partial dependency. For example if  $(A, B) \rightarrow (C, D)$  and  $B \rightarrow C$  and  $(A, B)$  is the primary key, then the functional dependence  $B \rightarrow C$  is a partial dependency.

**Second Normal Form (2NF)**

A relation is said to be in 2NF if,

- The relation "R" is in 1NF.
- There is no partial dependency exists in the relation "R".

In the following table partial dependencies exists. That is why it is in 1NF but not in 2NF.

Product_details:			
Cust_id	Cust_name	Prod_id	Prod_name
101	Ravi	P01	Pencil
102	Dinesh	P02	Eraser

**Prime attributes:** Cust\_id, Prod\_id

**FD:** Cust\_id → Cust\_name, Prod\_id → Prod\_name

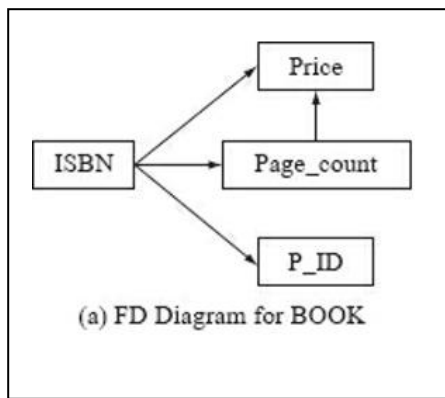
To achieve the desired 2NF we have to decompose it into two relations named as Customer and Product as follows:

Customer:		
Cust_id	Cust_name	Prod_id
101	Ravi	P01
102	Dinesh	P02

Product:	
Prod_id	Prod_name
P01	Pencil
P02	Eraser

### What is transitive dependency?

An attribute Y of a relation schema R is said to be transitively dependent on attribute X ( $X \rightarrow Y$ ), if there is a set of attributes A that is neither a candidate key nor a subset of any key of R and both  $X \rightarrow A$  and  $A \rightarrow Y$  hold. For example, the dependency ISBN → Price is transitive through Page\_count in relation schema BOOK (see the following Figure). This is because both the dependencies ISBN → Page\_count and Page\_count → Price hold and Page\_count is neither a candidate key nor a subset of the candidate key of BOOK.



### Third Normal Form

A relation “R” is in 3NF if,

- The relation “R” is in 2NF.
- For non-trivial FD  $X \rightarrow Y$ :
  - X is a super key or candidate key.
  - Y is a prime attribute.
- There is no transitive dependency exists.

For example, the following relation Employee is in 2NF but not in 3NF.

<b>Employee:</b>			
Emp_id	Emp_name	City	Zip
E01	Ravi	Kolkata	700001
E02	Dinesh	Mumbai	400001

**Transitive dependency:  $\text{Emp\_id} \rightarrow \text{Zip} \rightarrow \text{City}$**

**Also,  $\text{Zip} \rightarrow \text{City}$  where ‘Zip’ is not a super key and ‘City’ is not a prime attribute**

We can achieve the normalization at 3NF if decompose the relation as follows:

**Employee\_data:**

Emp_id	Emp_name	Zip
E01	Ravi	700001
E02	Dinesh	400001

**Zip\_details:**

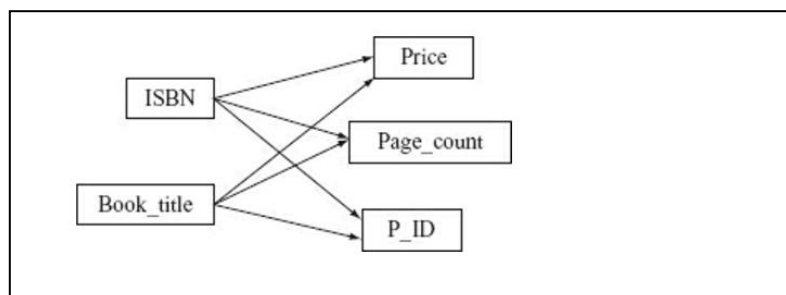
Zip	City
700001	Kolkata
400001	Mumbai

**Define the Boyce-Codd normal form. Give an example also.**

The Boyce-Codd normal form was proposed to deal with the relations having two or more candidate keys that are composite and that overlap. Definition: A relation schema  $R$  is in a Boyce-Codd normal form (BCNF) if, for every FD  $X \rightarrow A$  in  $F$ , where  $X$  is the subset of the attributes of  $R$ , and  $A$  is an attribute of  $R$ , one of the following statements holds:  $X \rightarrow Y$  is a trivial FD, that is,  $Y \subseteq X$ .  $X$  is a super key.

In simple terms, it can be stated as: A relation schema  $R$  is in BCNF if and only if every non-trivial FD has a candidate key as its determinant.

For example, consider a relation schema  $BOOK(ISBN, Book\_title, Price, Page\_count)$  with two candidate keys, namely,  $ISBN$  and  $Book\_title$ . The FD diagram for this relation schema is shown in the following Figure.



As shown in Figure 6.10, there are two determinants, namely  $ISBN$  and  $Book\_title$  and both of them are candidate keys. Thus, this relation schema is in BCNF.

### **Why is BCNF considered simpler as well as stronger than 3NF?**

BCNF is the simpler form of 3NF as it makes explicit reference to neither the first and second normal forms nor to the concept of transitive dependence. In addition, it is stronger than 3NF as every relation that is in BCNF is also in 3NF but the vice versa is not necessarily true. For example, the relation schema BOOK (described in 8), which is in BCNF, does not include any transitive dependency and thus, is in 3NF as well.

Now, consider another relation schema BOOK\_RATING(ISBN, Book\_title, R\_ID, Rating). Assuming that book titles are also unique, the relation has two candidate keys ( ISBN, R\_ID) and (Book\_title, R\_ID). This relation schema is not in BCNF since it contains two determinants, namely ISBN and Book\_title, which determine each other and neither of them is a candidate key. However, this relation is in 3NF.

### **What is multi-valued dependency? What is the difference between functional dependency and multi-valued dependency?**

Multi-valued dependencies (MVDs) are the generalization of functional dependencies. Definition: In a relation schema R, an attribute Y is said to be multi-dependent on attribute X( $X \twoheadrightarrow Y$ ) if and only if for a particular value of X, the set of values of Y is completely determined by the value of X alone and is independent of the values of Z where X, Y and Z are the subsets of the attributes of R.  $X \twoheadrightarrow Y$  is read as Y is multi-dependent on X or X multi-determines Y. Let us understand the notion of multi-valued dependencies with the help of an example. Consider the relation BOOK\_AUTHOR\_DETAIL as shown in the figure.

In this relation, each ISBN has a well-defined set of corresponding A\_ID. Similarly, for a particular A\_ID, a well-defined set of Phone exists. In addition, book is independent of the phone numbers of authors, due to which there is a lot of redundancy in this relation. Thus, the multi-valued dependencies that hold in this relation can be represented as follows:

$ISBN \twoheadrightarrow A\_ID$

$A\_ID \twoheadrightarrow Phone$

<u>ISBN</u>	<u>A_ID</u>	<u>Phone</u>
002-678-980-4	A002	376045
002-678-980-4	A008	765490
004-765-409-5	A008	765490

<u>ISBN</u>	<u>A_ID</u>	<u>Phone</u>
001-987-760-9	A001	923673
001-987-760-9	A001	923743
001-987-760-9	A003	419456
001-987-760-9	A003	419562
001-354-921-1	A005	678654
001-354-921-1	A005	678655
001-354-921-1	A005	678657

**BOOK\_AUTHOR\_DETAIL**

Functional dependencies prevent the existence of certain tuples in a relation. For example, if an FD  $X \rightarrow Y$  holds in R, then any  $r(R)$  cannot have two tuples having the same X value but different Y values. On the contrary, multi-valued dependencies do not rule out the presence of those tuples, rather, they require the presence of other tuples of a certain form in the relation.

#### **Fourth Normal Form (4NF)**

A relation is in **fourth normal form (4NF)** if it is in BCNF and contains no multi-valued dependencies.

#### **Fifth Normal Form (5NF)**

A table is in fifth normal form (5NF) or **Project-Join Normal Form (PJNF)** if it is in 4NF and it cannot have a lossless decomposition into any number of smaller tables.

#### **Domain-Key Normal Form (DKNF) or Sixth Normal Form (6NF)**

A table is in sixth normal form (6NF) or Domain-Key normal form (DKNF) if it is in 5NF and if all constraints and dependencies that should hold on the relation can be enforced simply by enforcing the domain constraints and the key constraints specified on the relation.

The domain-key normal form (DKNF) is a theoretical statement of how relationships are formed between tables.

.....

**What is SQL? What are the two major categories of SQL commands? Explain them.**

SQL stands for structured query language. It is a language that can be used for retrieval and management of data stored in relational database. It is a non-procedural language as it specifies what is to be retrieved rather than how to retrieve it. It can be used for defining the structure of data, modifying data in the database and specifying the security constraints. The two major categories of SQL commands are Data Definition Language (DDL) and Data Manipulation Language (DML). DDL provides commands that can be used to create, modify and delete database objects. DML provides commands that can be used to access and manipulate the data, that is, to retrieve, insert, delete and update data in a database.

**What is data type? What are the various data types supported by standard SQL?**

Data type identifies the type of data to be stored in an attribute of a relation and also specifies associated operations for handling the data. The common data types supported by standard SQL are as follows:

- **NUMERIC(p, s):** used to represent data as floating-point number. The number can have p significant digits (including sign) and s number of the p digits can be present on the right of decimal point. For example, the data type specified as NUMERIC(5, 2) indicates that the value of an attribute can be of form 332.32 .
- **INT or INTEGER:** used to represent data as a number without a decimal point.
- **SMALLINT:** used to represent data as a number without a decimal point. It is a subset of the INTEGER; so the default size is usually smaller than INT.
- **CHAR(n) or CHARACTER(n):** used to represent data as a fixed-length string of characters of size n. In case of fixed-length strings, a shorter string is padded with blank characters to the right. For example, if the value ABC is to be stored for an attribute with data type CHAR(8), the string is padded with five blanks to the right.
- **VARCHAR(n) or CHARACTER VARYING:** used to represent data as a variable length string of characters of maximum size n. In case of variable length string, a shorter string is not padded with blank characters.
- **DATE and TIME:** used to represent data as date or time. The DATE data type has three components, namely year, month and day in the form YYYY-MM-DD. The TIME data type also has three components, namely hours, minutes and seconds in the form HH:MM:SS.
- **BOOLEAN:** used to represent the third value unknown, in addition to true and false values, because of the presence of null values in SQL.
- **TIMESTAMP:** used to represent data consisting of both date and time. The TIMESTAMP data type has six components, year, month, day, hour, minute and second in the form YYYY-MM-DDHH: MM:SS[.sF], where F is the fractional part of the second value.



## Characteristics of SQL

The following are the important characteristics of SQL.

1. SQL is extremely flexible.
2. SQL uses a free form syntax that gives the user the ability to structure SQL statements in a way best suited.
3. It is a free formatted language, *i.e.*, there is no need to start SQL statements in a particular column or to be finished in a single line.
4. It has relatively few commands.
5. It is a non-procedural language.

## Advantages of SQL

The advantages of SQL are as follows:

1. SQL is a high level language that provides a greater degree of abstraction than procedural languages. The programmer has to specify what data is needed but need not to specify, how to retrieve it.
2. SQL is a unified language. The same language can be used to define data structures, querying data, control access to the data, insert, delete and modify occurrences of the data and so on.
3. All the programs written in SQL are portable, thus they can be moved from one database to another with very little modification. Such porting could be required when DBMS needs to be upgraded or changed.
4. The language is simple and easy to learn. It can handle complex situations very efficiently.
5. The language has sound theoretical base and there is no ambiguity about the way a query will interpret the data and produce the results. Thus the results to be expected are well defined.
6. SQL processes sets-of-records rather than just one record-at-a time. This set-at-a time feature of the SQL makes it more powerful.
7. SQL as a language is independent of the way it is implemented internally. This is because SQL specifies what is required and not how it should be done.
8. SQL enables its users to deal with a number of database management systems where it is available.

## What do you mean by Embedded and Dynamic SQL?

- Embedded SQL defines the way the SQL statements can be embedded within general purpose programming languages like C, C++, Cobol, Pascal etc. The language in which SQL queries are embedded is referred to as a **host language**. The SQL queries embedded in the host language constitute embedded SQL.
- Dynamic SQL allows programs to construct and submit SQL queries at run time.

Consider the following two relations:

Emp (Employee)						
EID	Name	Salary	Hire-date	Job	DID	MID
701	Deepak	8000	5-Jan-2001	Analyst	30	707
702	Naresh	9000	10-Jan-2001	Manager	10	707
703	Sumesh	7000	5-Feb-2001	Salesman	20	705
704	Aditya	9000	27-Nov-2003	Analyst	30	707
705	Lalit	6500	8-Oct-2002	Manager	20	707
706	Amit		4-Nov-2004	Clerk	10	702
707	Vishal	9500	1-Jan-2001	Manager	30	
708	Sumit	8000	6-Jan-2006	Accountant	10	702

Dept (Department)			
DID	DName	Loc	MID
10	Accounts	Bangalore	702
20	Sales	Delhi	705
30	Research	Pune	707
40	Developing	Hyderabad	

## Data Manipulation in SQL

SQL has one basic statement for retrieving information from the database: The SELECT statement. SQL also provides **Three** other DML statements to modify the database. These statements are: **update**, **delete** and **insert**.

### Select Statement

Select statement is used to retrieve information from table.

**Syntax :** select < column list > from < table name >.

**Example :** To display all department ID and the Department name, the query is

***select DID, DName from Dept ;***

Output:

DID	DName
10	Accounts
20	Sales
30	Research
40	Developing

**Example :** To select all columns use “\*”.

***select \* from Dept;***

Output:

DID	DName	Loc	Manager-ID
10	Accounts	Bangalore	702
20	Sales	Delhi	705
30	Research	Pune	707
40	Developing	Hyderabad	

**Column Alias :** You can give name to columns of your choice by using keyword “As” (gives column name in upper-case letter) or by using “ ” (gives column name as specified in query).

**Example :** *select DID As Department\_ID, DName from Dept ;*

Output:

Department-ID	DName
10	Accounts
20	Sales
30	Research
40	Developing

**Eliminating Duplicate Rows :** To eliminate duplicate rows, the keyword ‘DISTINCT’ is used.

**Example :** i) *select salary from Emp;*

ii) *select DISTINCT salary from Emp;*

Output:

Salary	Salary
8000	8000
9000	9000
7000	7000
9000	6500
6500	9500
9500	
8000	
(i)	(ii)

**Arithmetic Operators and NULL Values :** SQL provides arithmetic operators to perform calculations. Arithmetic operators with their precedence are

Description	Operator
Multiply	*
Divide	/
Add	+
Subtract	-

A null value is unknown value and it is different from zero. Any arithmetic operation with null value gives null results.

**Example :** Suppose you want to see increased salary of each employee by 500.

***select EID, salary + 500 "New Salary" from Emp;***

Output:

EID	New salary
701	8500
702	9500
703	7500
704	9500
705	7000
706	
707	10000
708	8500

**Where Clause :**  
particular rows

WHERE clause is used to select from table.

**Syntax :** select <column list> from <table name> where <condition>.

**Example :** List the name of employees having salary 9000.

***select name from emp where salary = 9000;***

Output:

Name
Naresh
Aditya

**Comparison or relational operators :** The relational operators provided by SQL are as follows.

Description	Operator
Equal to	=
Greater than	>
Less than	<
Greater that equal to	>=

Less than equal to	<=
Not equal to	<>

**Example :** List the name of employees having salary not equal to `9000.

***select name from emp where salary <> 9000;***

Output:

Name
Deepak
Sumesh
Lalit
Amit
Vishal
Sumit

**Special operators :** Special operators provided by SQL are as follows :

Description	Operator
Checking the value within a set	IN
Checking the value within a range	BETWEEN
Matching the pattern of characters	LIKE
Checking the null value	IS NULL

**Example:** List the EID and names of employees who were hired by company from 5–Feb–2001 to 1–Jan–2006.

***SELECT EID, Name FROM Emp WHERE Hire\_Date in (5-Feb-2001, 1-Jan-2006);***

Output:

EID	Name
703	Sumesh
704	Aditya
705	Lalit
706	Amit

**Example:** List the EID and names of employees having MID equal to null.

***select EID, name from Emp where MID IS NULL;***

Output:

EID	Name
707	Vishal

**Two** symbols with LIKE operator can be used:

(i) % → It represents any sequence of zero or more characters.

(ii) \_ (underscore) → It represents any single character.

**Example:** List the names of employees ending with 'it'.

***select name from Emp where name LIKE '% it';***

Output:

Name
Lalit
Amit
Sumit

**Example:** List the names of employees having second alphabet of their names is 'a'.

***select name from Emp where name LIKE '\_ a %';***

Output:

Name
Naresh
Lalit

**Logical operators:** Logical operators are used to combine two conditions. Following are the logical operators provided by SQL.

Description	Operator
It returns true if both conditions are true	AND
It returns true if either condition is true	OR
It returns true if condition is true	NOT

**Example:** List name of employees having salary less than ` 8500 and MID is 707.

***select name from Emp where salary <= 8500 AND MID = 707;***

Output:

Name
Deepak
Lalit

**Order by Clause :** Order by clause is used to sort rows in both ascending and descending order.

1. **ASC :** To sort rows in ascending order (By default).
2. **DESC :** To sort rows in descending order.

**Syntax :** select <column list> from <table name> where <condition> order by <column list> <ASC/DESC>;

**Example:** List name of employees in ascending order.

***select name from Emp order by name ;***

Output:

Name
Aditya
Amit
Deepak
Lalit
Naresh
Sumesh
Sumit
Vishal

## Functions in SQL

Functions are used to manipulate data but these are more powerful than simple queries.

### ***Types of Functions:***

1. Single row functions
2. Group functions.

Single row functions are further divided into:

1. Character Functions
2. Arithmetic Functions
3. Date Functions
4. Conversion Functions

## 5. General Functions.

**Dual table :** Dual table is used to explain single row functions. It has one column name Dummy and one row having value x.

Function	Description
LENGTH('string')	It returns the number of characters in string.
LOWER('string')	It converts string to lowercase.
UPPER('string')	It converts string to uppercase.
INITCAP('string')	It converts only first character of each word in string to uppercase.
CHR(x)	It returns equivalent character value of integer x.
CONCAT('string 1', 'string 2')	It joins both the strings. It is equivalent to concatenation operator.
REPLACE('string', 'searchstr', 'replace str')	It replaces every occurrence of search str with replace str within string.
SUBSTR('string', m[, n])	It returns a substring starting from mth position and upto n <sup>th</sup> position.
INSTR('string', 'char')	It returns position of first occurrence of char in string.
LPAD('string', n, 'char')	It left justified the string and fill remaining positions with char to a total width of n.
RPAD('string', n, 'char')	It right justified the string and fill remaining positions with char to a total width of n.
LTRIM('char' FROM 'string')	It trims leading char from string.
RTRIM('char' FROM 'string')	It trims trailing char from string.

**Example :** select LENGTH('Vivek') "Output" from dual;

Output
5

select REPLACE('Amit and Sumit', 'mit', 'zi') "Output" from dual;

Output
Azi and Suzi

select RPAD (name, 10, '\*\*') "Emp\_name", LENGTH (JOB) from Emp;

Emp_name	Length (JOB)
****Deepak	7
****Naresh	7
****Sumesh	8
****Aditya	7
*****Lalit	7
*****Amit	5
****Vishal	7
*****Sumit	10



**Number Functions :** Number functions are also known as arithmetic functions. They accept numeric data and returns numeric values.

Function	Description
CEIL(x)	It returns the smallest integer greater than or equal to x.
FLOOR(x)	It returns the largest integer less than or equal to x.
ABS(x)	It returns the absolute value of x.
Power(x, y)	It returns x raised to the power y.
Mod(x, y)	It returns the remainder of x divided by y.
SIGN(x)	It returns +1 if x is positive or -1 if x is negative.
ROUND(x, y)	It rounds the column. y specify the number of digits after decimal. If y is omitted then there are no decimal places. If y is negative, numbers to the left of the decimal point are rounded.
Exp(x)	It returns e raised to the power x.
SQRT(x)	It returns square root of x. If x is negative NULL is returned.
TRUNC(x, y)	It truncates the column. y specify the number of digits after decimal. If y is omitted then there are no decimal places. If y is negative, numbers to the left of the decimal point are truncated.

**Example:** select CEIL(77.7) from dual;

CEIL(77.7)
78

select FLOOR(69.2) "Output" from dual;

Output
69

*select ABS(-19) "Output" from dual;*

Output
19

*select SIGN(-9), SIGN(8) from dual;*

Output
-1

Output
+1

**Date Functions** : Date functions accept Date data type input and returns Date data type except MONTHS\_BETWEEN function. By default, date format in oracle is DD-MON-RR (12-Nov-81).

Date Functions	
Function	Description
SYSDATE	It returns the system date.
NEXT_DAY('date', 'day')	It returns the date of next specified day of the week after the 'date'.
LAST_DAY('date')	It returns the date of last day of the month.
ADD_MONTHS('date', n)	Add n months to 'date'.
MONTHS_BETWEEN('date 1', 'date 2')	It returns the number of months between 'date 1' and 'date 2'.
ROUND(d [, format])	It returns date rounded to the specified format. Default format is 'DD'.
TRUNC(d [, format])	It returns date truncated to the specified format. Default format is 'DD'.

**Example** : *select SYSDATE from dual;*

SYSDATE
21-FEB-19

**Conversion Functions** : Conversion functions are used to convert one data type into other data type.

Function	Description
TO_CHAR('date', 'F')	It converts 'date' into character format 'F'.
TO_DATE('char', 'F')	It converts string ('char' in date format) into date format 'F'

**Example:** *select TO\_CHAR('15-Nov-1988', 'MONTH') from Dual;*

TO_CHAR
NOVEMBER

*select TO\_DATE('DECEMBER', 'MM') from dual;*

TO_DATE
12

**General Functions :** General functions can accept any data type as input and pertain to the use of NULL values.

Function	Description
USER	It returns the name of the current user.
NVL(expr 1, expr 2)	It converts NULL value given by expr 1 to value given by expr 2
NVL2(expr 1, expr 2, expr 3)	It returns expr 2 if expr 1 is NULL otherwise it returns expr 3
UID	It returns the integer value which uniquely identify the oracle user.
NULLIF(expr 1, expr 2)	It compares expr 1, expr 2 and returns NULL if they are equal otherwise it returns expr 1.
COALESCE(expr 1, expr 2 ..., expr N)	It returns first non-null expression.

**Example:** *select USER from dual;*

USER
SCOTT

## Joining of Tables

If we need information from more than one table then we use joins. To join tables the condition need to be specified.

**Cartesian Product :** In Cartesian product there is no join condition. It returns all possible combinations of rows.

**Example:** select EID, LOC from Emp, Dept;

EID	LOC	8 × 4 = 32 row
701	Bangalore	
701	Delhi	
-	-	
-	-	
708	Hyderabad	

*Syntax of Join: The following is the common syntax for all types of JOIN.*

*select table 1.columns, table 2.columns from table 1, table 2  
where table 1.column N = table 2.column M;*

**Equijoin :** When two or more tables are joined by equality of values in one or more columns then it is called Equijoin. More than two tables can be joined by using logical operators.

**Example:** Display EID and DName of all employees by joining over DID.

**select Emp.EID, Dept.DName from Emp, Dept where Emp.DID = Dept.DID**

Output:

EID	DName
701	Research
702	Accounts
703	Sales
704	Research
705	Sales
706	Accounts
707	Research
708	Accounts

**Outer Join :** The outer join operator is '(+)'. During simple joining some rows are missing due to null value. To display these rows use outer join operator towards deficient side.

**Example:** Display EID and DName of employees by joining over MID.

SELECT e.EID, d.DName FROM Emp e, Dept d WHERE e.MID = d.MID;	SELECT e.EID, d.DName FROM Emp e, Dept d WHERE e.MID (+) = d.MID;	SELECT e.EID, d.DName FROM Emp e, Dept d WHERE e.MID = d.MID (+);																																																				
<table><tr><th>EID</th><th>DName</th></tr><tr><td>701</td><td>Research</td></tr><tr><td>702</td><td>Accounts</td></tr><tr><td>703</td><td>Sales</td></tr><tr><td>704</td><td>Research</td></tr><tr><td>705</td><td>Sales</td></tr><tr><td>706</td><td>Accounts</td></tr><tr><td>708</td><td>Accounts</td></tr></table>	EID	DName	701	Research	702	Accounts	703	Sales	704	Research	705	Sales	706	Accounts	708	Accounts	<table><tr><th>EID</th><th>DName</th></tr><tr><td>701</td><td>Research</td></tr><tr><td>702</td><td>Accounts</td></tr><tr><td>703</td><td>Sales</td></tr><tr><td>704</td><td>Research</td></tr><tr><td>705</td><td>Sales</td></tr><tr><td>706</td><td>Accounts</td></tr><tr><td>707</td><td></td></tr><tr><td>708</td><td>Accounts</td></tr></table>	EID	DName	701	Research	702	Accounts	703	Sales	704	Research	705	Sales	706	Accounts	707		708	Accounts	<table><tr><th>EID</th><th>DName</th></tr><tr><td>701</td><td>Research</td></tr><tr><td>702</td><td>Accounts</td></tr><tr><td>703</td><td>Sales</td></tr><tr><td>704</td><td>Research</td></tr><tr><td>705</td><td>Sales</td></tr><tr><td>706</td><td>Accounts</td></tr><tr><td>708</td><td>Accounts</td></tr><tr><td></td><td>Developing</td></tr></table>	EID	DName	701	Research	702	Accounts	703	Sales	704	Research	705	Sales	706	Accounts	708	Accounts		Developing
EID	DName																																																					
701	Research																																																					
702	Accounts																																																					
703	Sales																																																					
704	Research																																																					
705	Sales																																																					
706	Accounts																																																					
708	Accounts																																																					
EID	DName																																																					
701	Research																																																					
702	Accounts																																																					
703	Sales																																																					
704	Research																																																					
705	Sales																																																					
706	Accounts																																																					
707																																																						
708	Accounts																																																					
EID	DName																																																					
701	Research																																																					
702	Accounts																																																					
703	Sales																																																					
704	Research																																																					
705	Sales																																																					
706	Accounts																																																					
708	Accounts																																																					
	Developing																																																					

**Self Join :** A table can be joined to itself by using self joins.

**Example:** Display the name of employees and name of their managers.

select e.Name "Employee", m.Name "Manager" from Emp e, Emp m where e.MID = m.MID;

Output:

Employee	Manager
Deepak	Vishal
Naresh	Vishal
Sumesh	Lalit
Aditya	Vishal
Lalit	Vishal
Amit	Naresh
Sumit	Naresh

## Group Functions

Group functions are those functions that operate on a group of rows and returns a single result per group. Group may be entire table or a part of table. All the group functions ignore null values.

Function	Description
MAX(column name)	It returns the maximum value of a given attribute, ignoring NULL values.
MIN(column name)	It returns the minimum value of a given attribute, ignoring NULL values.
AVG([DISTINCT/ALL] column name)	It returns the average value of column values, ignoring NULL values.
STDDEV([DISTINCT/ALL] column name)	It returns the standard deviation of column values, ignoring NULL values.
Sum([DISTINCT/ALL] column name)	It returns the sum of column values, ignoring NULL values.
COUNT(* [DISTINCT] ALL  column name)	It returns the total number of rows.
VARIANCE([DISTINCT/ALL] column name)	It returns the statistical variance, ignoring NULL values.

**Example:**

SELECT AVG(Salary), AVG(DISTINCT Salary) FROM mp;	
AVG(Salary)	AVG(DISTINCT Salary)
8142.8	5714.2

**Group by Clause :** The GROUP BY clause is used to divide table into groups. A group may contain whole of the table or a collection of rows.

**Syntax :** select <column name> from <table name> where <condition> GROUP BY <column name>;

Column alias cannot be used with group by clause.

**Example 44 :** Display job and average salary paid by company for a particular job.

**select Job, AVG(salary) from emp Group By Job;**

Output:

Job	AVG(Salary)
Accountant	8000
Analyst	8500
Manager	8333.33
Salesman	7000

Rows are sorted by ascending order according to the column specified in GROUP BY clause (By default). To display rows in order according to the user, ORDER BY clause can be used.

**HAVING Clause :** The HAVING clause to apply restrictions on group. Like Where clause is used to restrict single rows, Having clause is used to restrict group, (collection of rows).

**Syntax :** select <column name> from <table name> where <condition> Group By <column name> Having <group condition>

**Example 45 :** Display job and average salary paid by company for a particular job in descending order according to their average salary and average salary must be greater than 7500.

select Job, AVG(Salary) from Emp Group by Job Having AVG(Salary) > 7500 Order By AVG(Salary) DESC;

Output:

Job	AVG(Salary)
Analyst	8500
Manager	8333.33
Accountant	8000

## Insert Statement

Insert statement is used to insert or add new rows in table.

**Syntax :** INSERT INTO <table name> (column 1, column 2,....., column n) VALUES (value 1, value 2,....., value n);

- Only a single row is inserted at a time
- Name of columns can be given in any order.

**Example:** Insert new rows in Dept. table.

**Insert INTO Dept (DID, DName, Loc, MID) VALUES (10; 'Accounts'; 'Bangalore', 708);**

## Update Statement

Update statement is used to modify the values of existing rows.

**Syntax :** UPDATE <table name> SET <(column 1 = value 1), (column 2 = value 2),....., (column *n* = value *n*)> WHERE <condition>;

- All rows in the table satisfies the condition are updated.

**Example:** Update the MID of DName Accounts to 702

**UPDATE Dept SET MID = 702 WHERE DName = 'Accounts';**

## Delete Statement

Delete statement is used to remove existing rows from table.

**Syntax :** DELETE FROM <table name> WHERE <condition>;

**Example:** Delete row from Dept having DName = Testing

**DELETE FROM Dept WHERE DName = 'Testing';**

## Data Definition Language (DDL)

SQL DDL commands are used to create, modify or remove database structures including tables. These commands have an immediate effect on the database, and also record information in the data dictionary. The following examples show working of some of the DDL commands.

### Create Table

Create table statement is used to create new tables in database.

**Column Constraints :** Constraints are rules which are forced on database to follow them for consistency purpose.

Constraint	Description
Not Null	By using this constraint, null value to a particular attribute cannot be assigned.
Unique	Each value of an attribute must be unique
Primary Key	Value at each column must be unique and Not Null
Foreign Key	Particular attribute must follow referential integrity.
Check	Specified condition must be true for attribute



**Example:** Create a table, Dept, with attributes DID, DName, Loc and MID. DName must be unique.

```
CREATE TABLE Dept ( DID Number(4), DName Varchar2(20),  
Constraint dname-unique UNIQUE Loc Varchar2(20), MID Number(4));  
or  
CREATE TABLE Dept( DID Number(4), DName Varchar2(20), Loc Varchar2(20),  
MID Number(4), CONSTRAINT dname_unique UNIQUE (DName));
```

### Alter Table Statement

Alter table statement is used to add or drop a constraint. It can also be used to disable or enable any constraint. Alter table statement is also used to add or drop columns of tables and to modify name and attributes of an existing column.

#### Syntax:

- (i) **To add a constraint:** ALTER Table <table name> ADD CONSTRAINT <condition>;
- (ii) **To drop a constraint:** ALTER Table <table name> DROP CONSTRAINT <constraint name> CASCADE CONSTRAINTS;
- (iii) **To enable a constraint:** ALTER Table <table name> ENABLE CONSTRAINT <constraint name>;
- (iv) **To disable a constraint:** ALTER Table <table name> DISABLE CONSTRAINT <constraint name> CASCADE;
- (v) **To add new column:** ALTER TABLE <table name> ADD (<column name> <data type(size)>;
- (vi) **To drop a column:** ALTER TABLE <table name> DROP COLUMN <column name>;
- (vii) **To modify a column:** ALTER TABLE <table name> MODIFY (<column name> <new data type | new size | new default value>;

### Describe Statement

It describes the structure of table.

**Syntax :** DESCRIBE <table name>;

**Example 61 :** Describe the structure of table Emp.

**DESCRIBE Emp;**

Output:

Name	NULL	Type
EID	NOT NULL	NUMBER(4)
NAME		VARCHAR2(30)
SALARY	NOT NULL	NUMBER(6)
HIRE-DATE		DATE
JOB		VARCHAR2(20)
DID		NUMBER(4)
MID		NUMBER(4)

### Drop Statement

Drop table statement is used to remove table from database.

**Syntax :** DROP TABLE <table name>;

**Example:** Remove table student (Example-62) from database.

**DROP TABLE Student;**

### **Data Control Language (DCL)**

Data Control Language(DCL) is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges.

In DCL we have two commands,

- **GRANT:** Used to provide any user access privileges or other privileges for the database.
- **REVOKE:** Used to take back permissions from any user.

**Grant:** Grant statement is used to give different permissions on different portions of database to different users. In a multi-user database management system, it is required to grant different permissions for security purposes.

**Syntax :** GRANT <privilege-list>| ALL ON <object> TO <user-list>| PUBLIC [WITH GRANT OPTION]

**Ex.:** Grant all the permissions on table Emp to all users

**GRANT ALL ON Emp TO PUBLIC;**

**Ex.:** Grant ALTER authority on table Emp to user Nick with the capability to grant those authorities to other users.

**GRANT ALTER ON Emp TO Nick WITH GRANT OPTION;**

**REVOKE:** REVOKE statement is used to take away any authority from a user that was granted earlier.

**Syntax :** REVOKE <privilege-list>| ALL ON <table name> [(column-comma-list)] FROM <user-list>| PUBLIC

**Ex.:** REVOKE the UPDATE permission on table Dept from Rohan.

**REVOKE UPDATE ON Dept FROM Rohan;**

**Ex.:** REVOKE INSERT and UPDATE permission on Name and EID columns of table Emp from all users.

**REVOKE INSERT, UPDATE (Name, EID) ON Emp FROM PUBLIC;**

### **Transaction Control Language (TCL)**

Transaction Control Language can be defined as the portion of a database language used for maintaining consistency of the database and managing transactions in database.

There are three commands that come under the TCL: COMMIT, ROLLBACK, SAVEPOINT

**Commit:** A transaction is completed successfully after commit. Commit statement is used to make data changes permanent to database.

**Syntax :** COMMIT;

**Rollback:** Rollback statement is used to terminate current transaction and discarding all data changes pending due to that transaction.

**Syntax :** ROLLBACK;

**Savepoint:** It is used to partially commit the current transaction and put a savepoint at that position.

**Syntax :** SAVEPOINT <name>;

If second savepoint will be created within same transaction then earlier savepoint is automatically discarded by database.

.....

### **Define transaction. What are the properties of a transaction?**

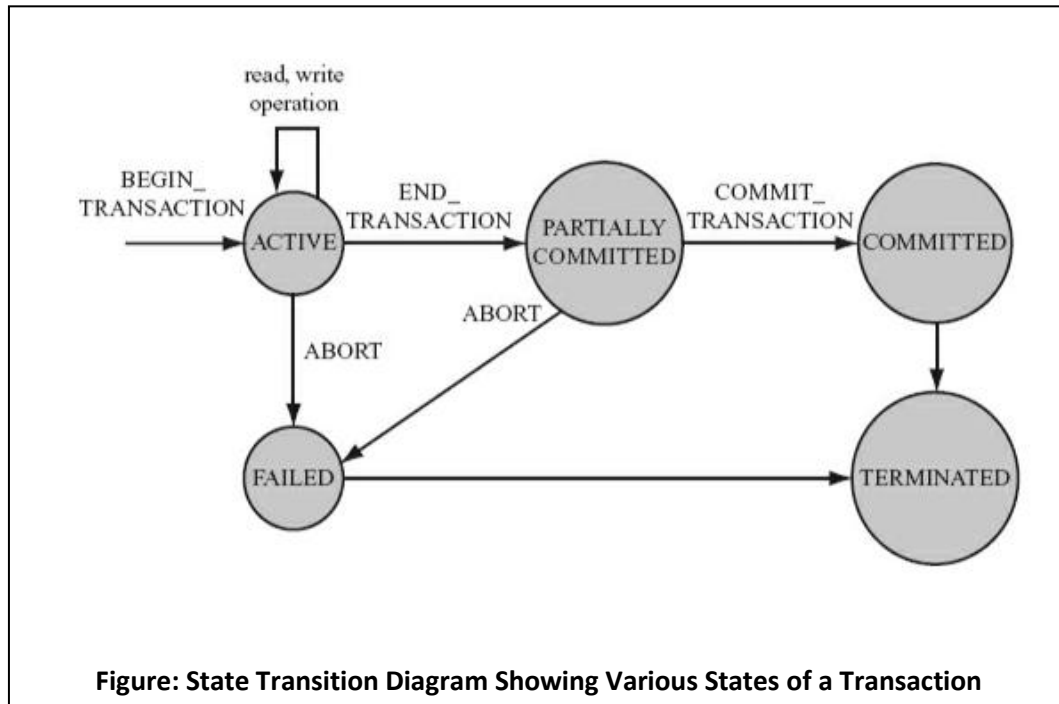
Explain with the help of an example. Ans: A collection of operations that form a single logical unit of work is called a transaction. The operations that make up a transaction typically consist of requests to access existing data, modify existing data, add new data or any combination of these requests. The statements of a transaction are enclosed within the begin transaction and end transaction statements. To ensure the integrity of the data, the database system must maintain some desirable properties of the transaction. These properties are known as

**ACID properties**, the acronym derived from the first letter of the terms atomicity, consistency, isolation and durability.

- **Atomicity** implies that either all of the operations that make up a transaction should execute or none of them should occur.
- **Consistency** implies that if all the operations of a transaction are executed completely, the database is transformed from one consistent state to another.
- **Isolation** implies that each transaction appears to run in isolation with other concurrently running transactions.
- **Durability** (also known as permanence) implies that once a transaction is completed successfully, the changes made by the transaction persist in the database, even if the system fails.

**Explain state transition diagram. Explain, when a transaction is said to be failed.**

State transition diagram is a diagram that describes how a transaction passes through various states during its execution. Whenever a transaction is submitted to a DBMS for execution, either it executes successfully or fails due to some reasons. During its execution, a transaction passes through various states that are active, partially committed, committed, failed and aborted as shown in Figure.



A transaction enters into the active state with its commencement. At this point, the system marks **BEGIN\_TRANSACTION** operation to specify the beginning of the transaction execution. During its execution, the transaction stays in the active state and executes several **READ** and **WRITE** operations on the database. The **READ** operation transfers a data item from the database to a local buffer of the transaction that has executed the read operation. The **WRITE** operation transfers the data item from the local buffer of the transaction back to the database.

Once the transaction executes its final operation, the system marks **END\_TRANSACTION** operation to specify the end of the transaction execution. At this point, the transaction enters into the partially committed state. The actual output at this point may still be residing in the main memory and, thus, any kind of hardware failure might prevent its successful completion. In such a case, the transaction may have to be aborted.

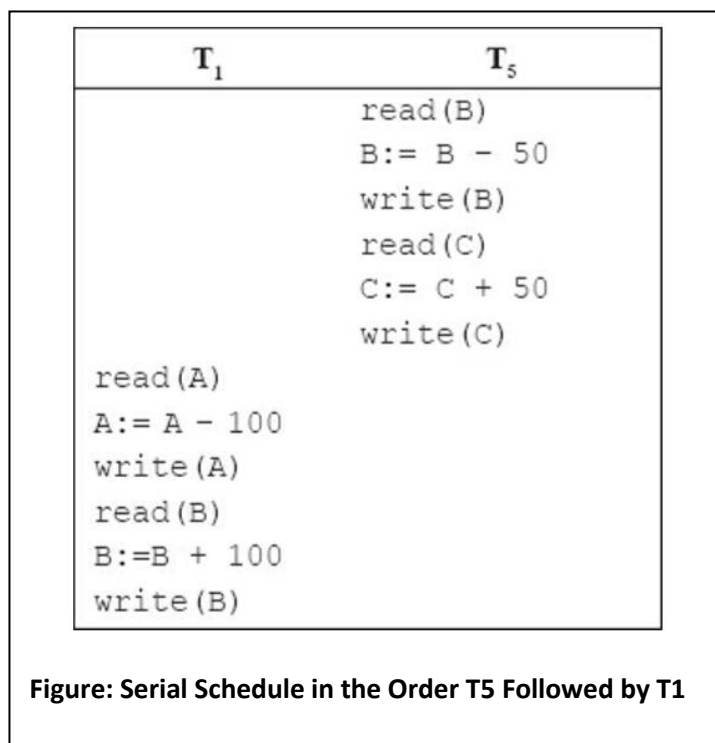
Before actually updating the database on the disk, the system first writes the details of updates performed by the transaction in the log file. The log file is then written to the disk so that, even in case of failure, the system can re-construct the updates performed by the transaction when the system restarts after the failure. When this information is successfully written out in the log file, the system

marks COMMIT\_TRANSACTION operation to indicate the successful end of the transaction. Now, the transaction is said to be committed and all its changes must be reflected permanently in the database.

If the transaction is aborted during its active state or the system fails to write the changes in the log file, the transaction enters the failed state. The failed transaction must be rolled back to undo its effects on the database to maintain the consistency of the database. When the transaction leaves the system, it enters into the terminated state. At this point, the transaction information maintained in the log file during its execution is removed.

**Define serial schedule. Why is it always considered to be correct?**

Serial schedule is a schedule consisting of a sequence of instructions from various transactions, where the operations of one single transaction appear together in that schedule. Each transaction in a serial schedule is executed independently without any interference from the operations of other transactions. As long as every transaction is executed from beginning to end without any interference from other transactions, it gives a correct end result on the database. Therefore, every serial schedule is considered to be correct. Hence, for a set of  $n$  transactions,  $n!$  different valid serial schedules are possible. The serial schedule of transactions  $T_1$  and  $T_5$  in the order  $T_5$  followed by  $T_1$  is shown in Figure.



**Explain serializable schedule by giving an example.**

The consistency of the database under concurrent execution can be ensured by interleaving the operations of transactions in such a way that the final output is the same as that of some serial schedule of those transactions. Such a schedule is referred to as serializable schedule. Thus, a schedule  $S$  of  $n$  transactions  $T_1, T_2, T_3, \dots, T_n$  is serializable if it is equivalent to some serial schedule of the same  $n$  transactions.

Following figure shows a non-serial schedule of transactions T<sub>1</sub> and T<sub>5</sub>, which is equivalent to serial schedule shown in previous figure. After the execution of this schedule, the final values of accounts A, B and C are \$1900, \$1550 and \$550. Thus, the sum A + B + C is preserved and, hence, it is a serializable schedule.

T <sub>1</sub>	T <sub>5</sub>
read(A)	
A:=A - 100	
write(A)	
	read(B)
	B:= B - 50
	write(B)
read(B)	
B:=B + 100	
write(B)	
	read(C)
	C:=C + 50
	write(C)

**Figure: Serializable Schedule**

**What are result equivalent schedules? Explain with example.**

Two different schedules may produce the same final state of the database. Such schedules are known as result equivalent, since they have the same effect on the database. However, in some cases, two different schedules may accidentally produce the same final state of database. For example, consider two schedules S<sub>i</sub> and S<sub>j</sub> that are updating the data item Q, as shown in the following Figure. Suppose that the value of data item Q is \$100 then the final state of database produced by schedules S<sub>i</sub> and S<sub>j</sub> is the same, that is, they produce the same value of Q (\$200).

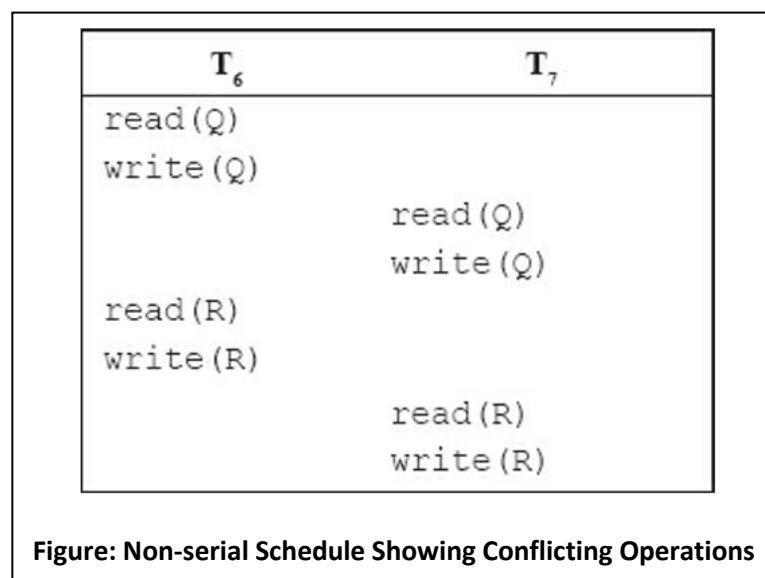
S <sub>i</sub>	S <sub>j</sub>
read(Q)	read(Q)
Q:= Q + 100	Q:= Q * 2
write(Q)	write(Q)

**Figure: Result Equivalent Schedules**

### Discuss the two different forms of schedule equivalence.

Two different forms of schedule equivalence are conflict equivalence and view equivalence that lead to the notions of conflict serializability and view serializability.

**Conflict equivalence and conflict serializability:** Two operations in a schedule are said to conflict if they belong to different transactions, access the same data item, and at least one of them is the write operation. On the other hand, two operations belonging to different transactions in a schedule do not conflict if both of them are read operations or both of them are accessing different data items. To understand the concept of conflicting operations in a schedule, consider two transactions T<sub>6</sub> and T<sub>7</sub> that are updating the data items Q and R in the database. A non-serial schedule of these transactions is shown in Figure below.



In the above Figure, the write(Q) operation of T<sub>6</sub> conflicts with the read(Q) operation of T<sub>7</sub> because both the operations are accessing the same data item Q, and one of these operation is the write operation. On the other hand, the write(Q) operation of T<sub>7</sub> is not conflicting with the read(R) operation of T<sub>6</sub>, because both are accessing different data items Q and R.

If a schedule S can be transformed into a schedule S' by performing a series of swaps of non-conflicting operations, then S and S' are conflict equivalents. Note that while swapping the order of execution of two conflicting operations cannot be changed because if they are applied in different order, they can have different effect on the database or on the other transactions in the schedule. Thus, two schedules are said to be conflict equivalent if the order of any two conflicting operations is same in both the schedules. A schedule, which is conflict equivalent to some serial schedule, is known as conflict serializable.

**View equivalence and view serializability:** Two schedules S and S' are said to be view equivalent if the schedules satisfy these conditions.

- The same set of transactions participates in S and S', and S and S' include the same set of operations of those transactions.
- If the transaction T<sub>i</sub> reads the initial value of a data item say Q in schedule S, then T<sub>i</sub> must read the initial value of same data item Q in schedule S' also.
- For each data item Q, if T<sub>i</sub> executes read(Q) operation after the write(Q) operation of transaction T<sub>j</sub> in schedule S, then T<sub>i</sub> must execute the read(Q) operation after the write(Q) operation of T<sub>j</sub> in schedule S' also.
- If the transaction T<sub>i</sub> performs the final write operation on any data item say Q in schedule S, then it must perform final write operation on the same data item Q in schedule S' also.

A schedule S is said to be view serializable if it is view equivalent to some serial schedule.

### **What is concurrency control? How is it implemented?**

When several transactions execute concurrently, they may result in interleaved operations, and the isolation property of transaction may no longer be preserved. Thus, it may leave the database in an inconsistent state. To understand, consider a situation in which two transactions concurrently access the same data item. One transaction modifies a tuple, and the other makes a decision on the basis of that modification. Now, suppose that the first transaction rolls back. At this point, the decision of the second transaction becomes invalid. Thus, there is a need to control the interaction among concurrent transactions, which is referred to as concurrency control.

The concurrency control is implemented with the help of concurrency control techniques, which ensure that the concurrent transactions maintain the integrity of a database by avoiding the interference among them and further ensure serializability order in the schedule of transactions. The various concurrency control techniques are locking, timestamp-based, optimistic and the multiversion technique.

### **Define lock. What are the two modes of locking?**

A lock is a variable associated with each data item that indicates whether a read or write operation can be applied to the data item. In addition, it synchronizes the concurrent access of the data item. Acquiring the lock by modifying its value is called locking. It controls the concurrent access and manipulation of the locked data item by other transactions and, hence, maintains the consistency and integrity of the database. Database systems mainly use two modes of locking, namely, exclusive locks and shared locks.

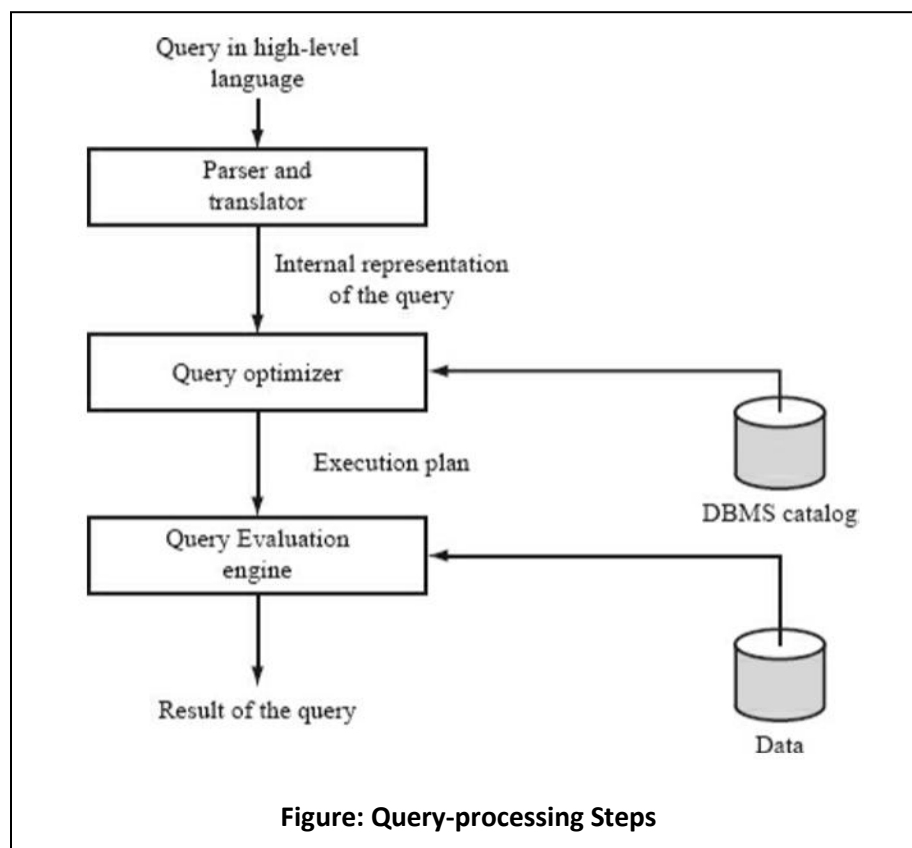
**Exclusive lock** (denoted by X) is the commonly used locking strategy that provides a transaction an exclusive control on the data item. A transaction that wants to read as well as write a data item must acquire an exclusive lock on the data item. Hence, an exclusive lock is also known as the update lock or write lock. If a transaction (say, T<sub>i</sub>) has acquired an exclusive lock on a data item (say, Q), no other transaction is allowed to access Q until T<sub>i</sub> releases its lock on Q.



**Shared lock** (denoted by S) can be acquired on a data item when a transaction wants to only read a data item and not modify it. Hence, it is also known as read lock. If a transaction  $T_i$  has acquired a shared lock on data item Q,  $T_i$  can read but cannot write on Q. In addition, any number of transactions can acquire shared locks on Q simultaneously. However, no transaction can acquire an exclusive lock on Q.

**What do you mean by query processing? What are the various steps involved in query processing? Explain with the help of a block diagram.**

Query processing includes translation of high-level queries into low-level expressions that can be used at the physical level of the file system, query optimization and actual execution of the query to get the result. It is a three-step process that consists of parsing and translation, optimization and execution of the query submitted by the user. These steps are discussed below:



- **Parsing and translation:** Whenever a user submits a query in high-level language (such as SQL) for execution, it is first translated into its internal representation suitable to the system. The internal representation of the query is based on the extended relational algebra. Thus, an SQL query is first translated into an equivalent extended relational algebra expression. During translation, the parser checks the syntax of the user's query according to the rules of the query language. It also verifies that all the attributes and relation names specified in the query are the valid names in the schema of the database being queried.

- **Optimization:** Generally, there are several possible ways of executing a query (known as execution strategies), and different execution strategies can have different costs. It is the responsibility of the query optimizer, a component of DBMS, to choose a least costly execution strategy. This process of choosing a suitable execution strategy for processing a query is known as query optimization. The metadata stored in the special tables called DBMS catalog is used to find the best way of evaluating a query.
- **Execution:** The chosen execution strategy is finally submitted to the query-evaluation engine for actual execution of the query to get the desired result.

**Define recovery manager. What are the properties of transactions that it preserves?**

The component of DBMS that is responsible for performing the recovery operations is called recovery manager. The main aim of recovery is to restore the database to the most recent consistent state. The recovery manager ensures that the two important properties of transactions namely, atomicity and durability are preserved. It preserves atomicity by undoing actions of uncommitted transactions and durability by ensuring that all the actions of committed transactions survive any type of failure.