UNIT-1

What is a Database?

A database is a collection of related data which represents some aspect of the real world. A database system is designed to be built and populated with data for a certain task.

What is DBMS?

Database Management System (DBMS) is software for storing and retrieving users' data while considering appropriate security measures. It consists of a group of programs which manipulate the database. The DBMS accepts the request for data from an application and instructs the operating system to provide the specific data. In large systems, a DBMS helps users and other third-party software to store and retrieve data.

DBMS allows users to create their own databases as per their requirement. The term “DBMS” includes the user of the database and other application programs. It provides an interface between the data and the software application.

Example of a DBMS

This database is maintaining information concerning students, courses, and grades in a university environment. The database is organized as five files:

- The STUDENT file stores data of each student
- The COURSE file stores contain data on each course.
- The SECTION stores the information about sections in a particular course.
- The GRADE file stores the grades which students receive in the various sections
- The TUTOR file contains information about each professor.

To define a database system:

- We need to specify the structure of the records of each file by defining the different types of data elements to be stored in each record.
- We can also use a coding scheme to represent the values of a data item.
- Basically, your Database will have 5 tables with a foreign key defined amongst the various tables.

History of DBMS

Here, are the important landmarks from the history:

- 1960 - Charles Bachman designed first DBMS system
- 1970 - Codd introduced IBM'S Information Management System (IMS)
- 1976- Peter Chen coined and defined the Entity-relationship model also known as the ER model
- 1980 - Relational Model becomes a widely accepted database component
- 1985- Object-oriented DBMS develops.
- 1990s- Incorporation of object-orientation in relational DBMS.
- 1991- Microsoft ships MS access, a personal DBMS and that displaces all other personal DBMS products.
- 1995: First Internet database applications
- 1997: XML applied to database processing. Many vendors begin to integrate XML into DBMS products.

Characteristics of Database Management System

- Provides security and removes redundancy
- Self-describing nature of a database system
- Insulation between programs and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing
- DBMS allows entities and relations among them to form tables.
- It follows the ACID concept (Atomicity, Consistency, Isolation, and Durability).
- DBMS supports multi-user environment that allows users to access and manipulate data in parallel.

DBMS vs. Flat File

DBMS	**Flat File Management System**

Multi-user access	It does not support multi-user access
Design to fulfill the need for small and large businesses	It is only limited to smaller DBMS system.
Remove redundancy and Integrity	Redundancy and Integrity issues
Expensive. But in the long term Total Cost of Ownership is cheap	It's cheaper
Easy to implement complicated transactions	No support for complicated transactions

Users in a DBMS environment

Following, are the various category of users of a DBMS system

Component Name	**Task**
Application Programmers	The Application programmers write programs in various programming languages to interact with databases.
Database Administrators	Database Admin is responsible for managing the entire DBMS system. He/She is called Database admin or DBA.
End-Users	The end users are the people who interact with the database management system. They conduct various operations on database like retrieving, updating, deleting, etc.

Popular DBMS Software

Here, is the list of some popular DBMS system:

- MySQL
- Microsoft Access
- Oracle
- PostgreSQL
- dBASE

- FoxPro
- SQLite
- IBM DB2
- LibreOffice Base
- MariaDB
- Microsoft SQL Server etc.

Application of DBMS

Sector	Use of DBMS
Banking	For customer information, account activities, payments, deposits, loans, etc.
Airlines	For reservations and schedule information.
Universities	For student information, course registrations, colleges and grades.
Telecommunication	It helps to keep call records, monthly bills, maintaining balances, etc.
Finance	For storing information about stock, sales, and purchases of financial instruments like stocks and bonds.
Sales	Use for storing customer, product & sales information.
Manufacturing	It is used for the management of supply chain and for tracking production of items. Inventories status in warehouses.
HR Management	For information about employees, salaries, payroll, deduction, generation of paychecks, etc.

Types of DBMS

Four Types of DBMS systems are:

- Hierarchical database
- Network database
- Relational database
- Object-Oriented database

Hierarchical DBMS

In a Hierarchical database, model data is organized in a tree-like structure. Data is Stored Hierarchically (top down or bottom up) format. Data is represented using a parent-child relationship. In Hierarchical DBMS parent may have many children, but children have only one parent.

Network Model

The network database model allows each child to have multiple parents. It helps you to address the need to model more complex relationships like as the orders/parts many-to-many relationship. In this model, entities are organized in a graph which can be accessed through several paths.

Relational model

Relational DBMS is the most widely used DBMS model because it is one of the easiest. This model is based on normalizing data in the rows and columns of the tables. Relational model stored in fixed structures and manipulated using SQL.

Object-Oriented Model

In Object-oriented Model data stored in the form of objects. The structure which is called classes which display data within it. It defines a database as a collection of objects which stores both data member's values and operations.

Advantages of DBMS

- DBMS offers a variety of techniques to store & retrieve data
- DBMS serves as an efficient handler to balance the needs of multiple applications using the same data
- Uniform administration procedures for data
- Application programmers never exposed to details of data representation and storage.
- A DBMS uses various powerful functions to store and retrieve data efficiently.
- Offers Data Integrity and Security
- The DBMS implies integrity constraints to get a high level of protection against prohibited access to data.

- A DBMS schedules concurrent access to the data in such a manner that only one user can access the same data at a time
- Reduced Application Development Time

Disadvantage of DBMS

DBMS may offer plenty of advantages but, it has certain flaws-

- Cost of Hardware and Software of a DBMS is quite high which increases the budget of your organization.
- Most database management systems are often complex systems, so the training for users to use the DBMS is required.
- In some organizations, all data is integrated into a single database which can be damaged because of electric failure or database is corrupted on the storage media
- Uses of the same program at a time by many users sometimes lead to the loss of some data.
- DBMS can't perform sophisticated calculations

DBMS vs. File System

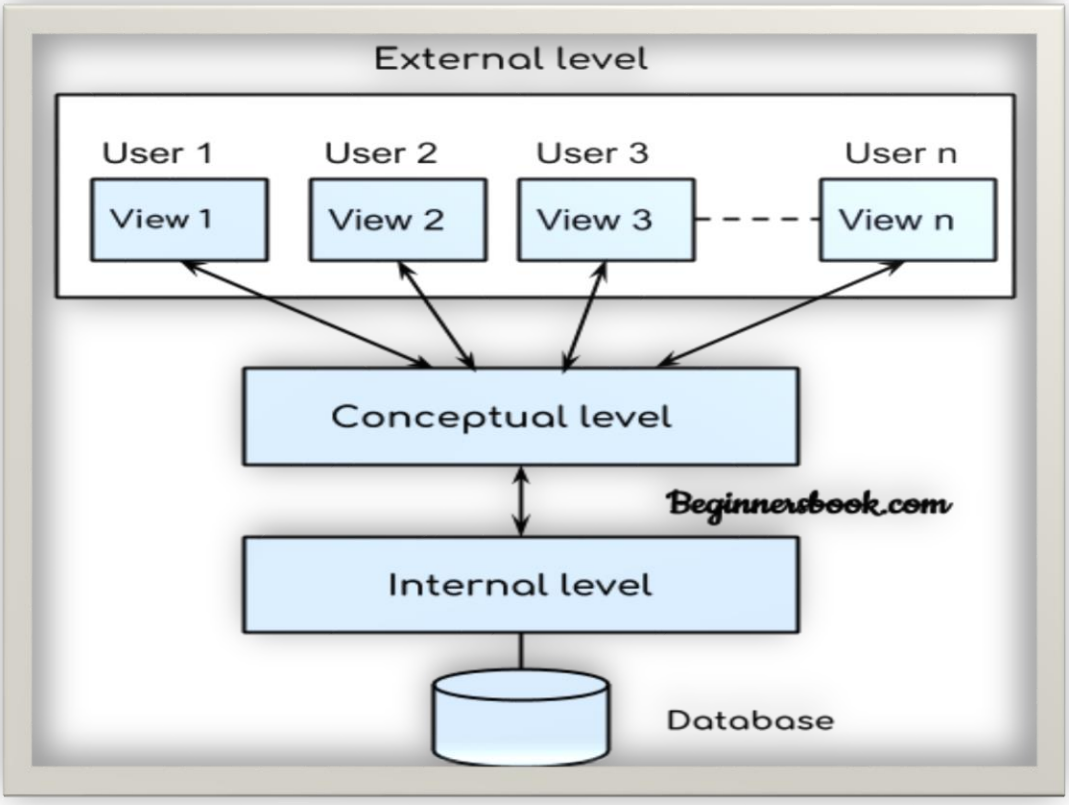
There are following differences between DBMS and File system:

DBMS	File System
DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	File system is a collection of data. In this system, the user has to write the procedures for managing the database.
DBMS gives an abstract view of data that hides the details.	File system provides the detail of the data representation and storage of data.
DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure.	File system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will lost.
DBMS provides a good protection mechanism.	It is very difficult to protect a file under the file system.

DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	File system can't efficiently store and retrieve the data.
DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access has many problems like redirecting the file while other deleting some information or updating some information.

DBMS Architecture

- The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
- The client/server architecture consists of many PCs and a workstation which are connected via the network.
- DBMS architecture depends upon how users are connected to the database to get their request done.



Types of DBMS Architecture

Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

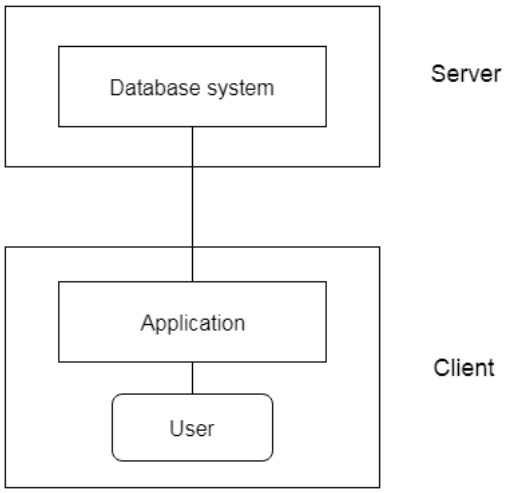
1-Tier Architecture

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.

- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

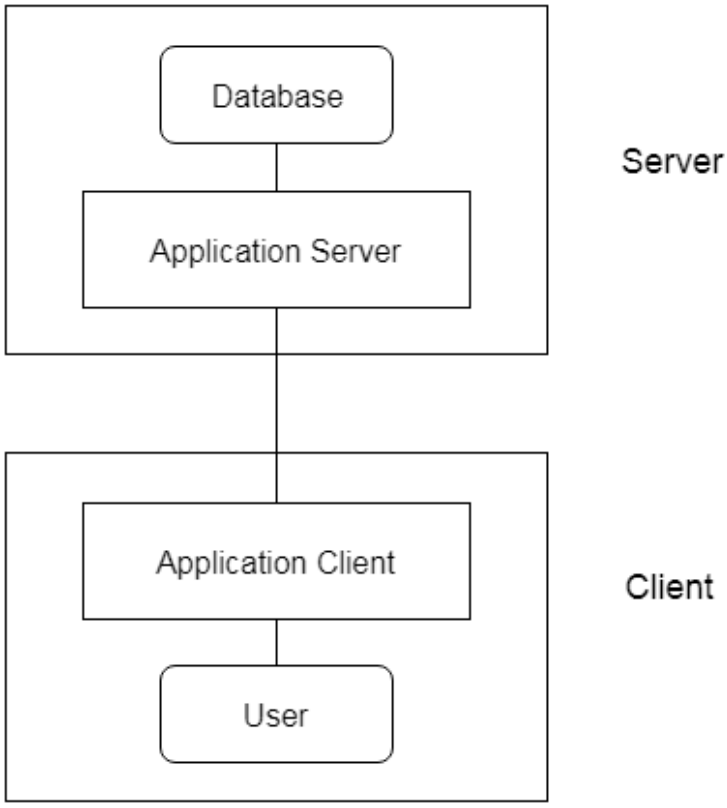
2-Tier Architecture

- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.



3-Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.



Instance and schema in DBMS Schema Definition of schema

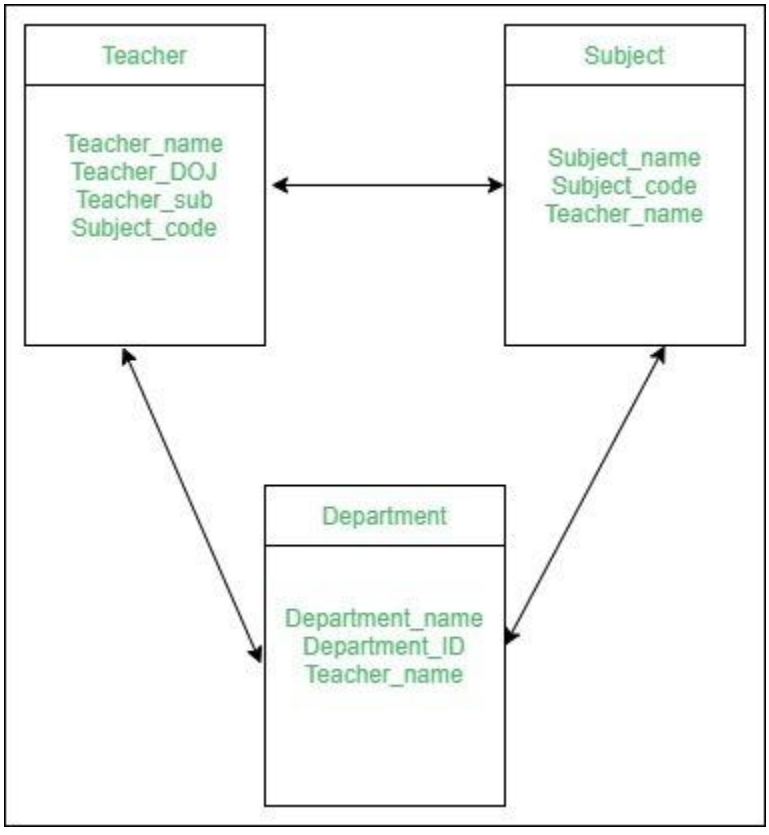
Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema. For example, we have a schema that shows the relationship between three tables: Course, Student and Section. The diagram only shows the design of the database, it doesn't show the data present in those tables. Schema is only a structural view (design) of a database as shown in the diagram below. The design of a database at physical level is called physical schema, how the data stored in blocks of storage is described at this level. Design of database at logical level is called logical schema, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level). Design of database at view level is called view schema. This generally describes end user interaction with database systems. DBMS Instance Definition of instance: The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable

declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database. For example, lets say we have a single table student in the database, today the table has 100 records, so today the instance of the database has 100 records. Lets say we are going to add another 100 records in this table by tomorrow so the instance of database tomorrow will have 200 records in table. In short, at a particular moment the data stored in database is called the instance that changes over time when we add or delete data from the database

Difference between Schema and Instance in DBMS

1. Instances:
Instances are the collection of information stored at a particular moment. The instances can be changed by certain CRUD operations as like addition, deletion of data. It may be noted that any search query will not make any kind of changes in the instances.
Example –
Let's say a table teacher in our database whose name is School, suppose the table has 50 records so the instance of the database has 50 records for now and tomorrow we are going to add another fifty records so tomorrow the instance have total 100 records. This is called an instance.

2. Schema:
Schema is the overall description of the database. The basic structure of how the data will be stored in the database is called schema.



Schema is of two types: Logical Schema, and Physical Schema.

1. **Logical Schema –** It describes the database designed at logical level.
2. **Physical Schema –** It describes the database designed at physical level.

Example –

Let's say a table teacher in our database name school; the teacher table requires the name, dob, doj in their table so we design a structure as:

```
Teacher table
name: String
doj: date
dob: date
```

Above given is the schema of the table teacher.

Difference between Schema and Instance:

SCHEMA	INSTANCE
It is the overall description of the database.	It is the collection of information stored in a database at a particular moment.
Schema is same for whole database.	Data in instances can be changed using addition, deletion, updating.
Does not change Frequently.	Changes Frequently.
Defines the basic structure of the database i.e. how the data will be stored in the database.	It is the set of Information stored at a particular time.

Difference between Schema and Database

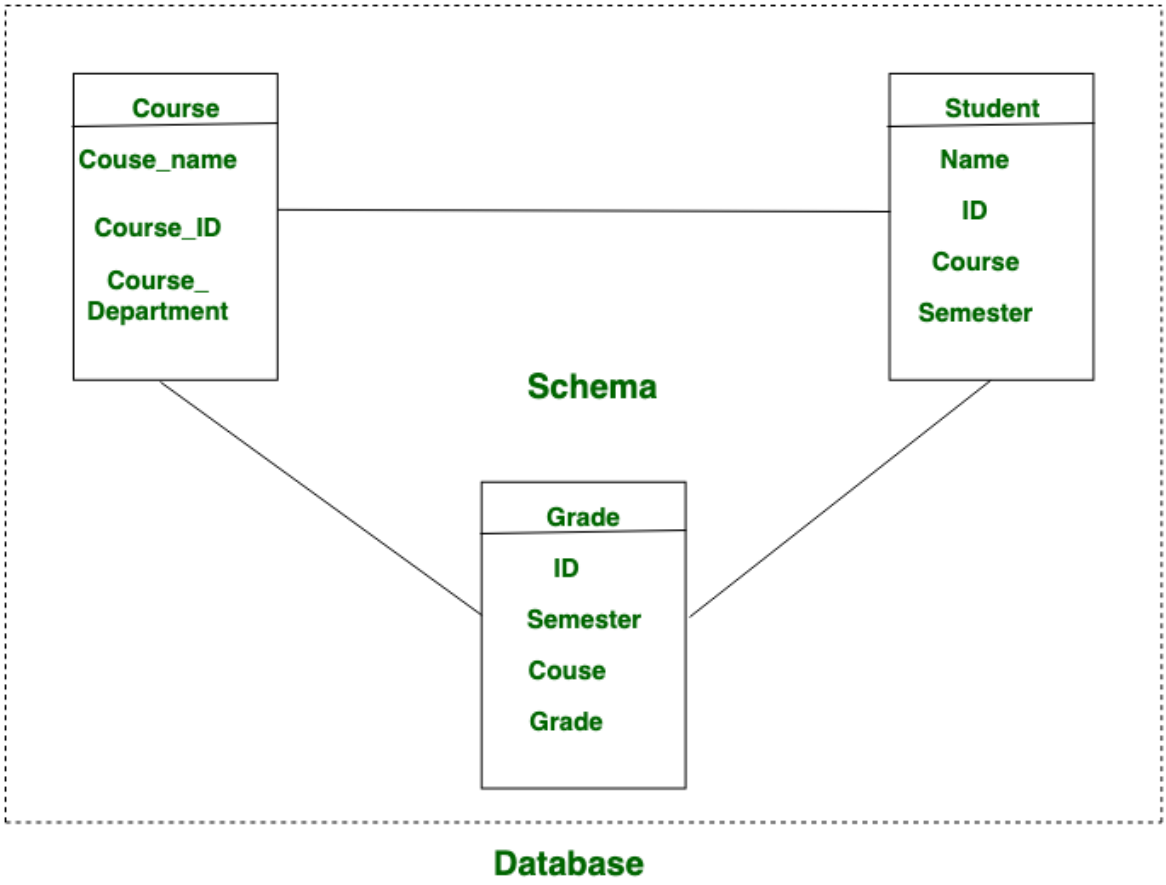
Last Updated: 24-09-2019

Database:

Database may be a common term in today's life. Several enterprises, firms, organization, institutes, etc. needs a system to store their information in a very well-formatted type in order that it might be simple to retrieve a helpful data out of it. whereas planning a information.

Schema:

Schema is such that describes the structural read of a information that confirms the tables that will be concerned in making a information, the table's attributes and their association.



Let's see the difference between Schema and Database:

S.NO	SCHEMA	DATABASE
1.	Schema may be a structural read of an info or database.	The info or database may be a assortment of reticulate knowledge.
2.	Schema once declared mustn't be changed often.	Knowledge during a info or database keeps on change all time, therefore database or info modifies often.
3.	In schema, Tables name, fields name, its sorts as well as	Database or info includes such schema, records, and constraints for the

	constraints are included.	information.
4.	For a info, Schema is specified by DDL.	In a database, The operations such as updates and ads are done using DML.

Data Independence

- Data independence can be explained using the three-schema architecture.
- Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

There are two types of data independence:

1. Logical Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.

2. Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.

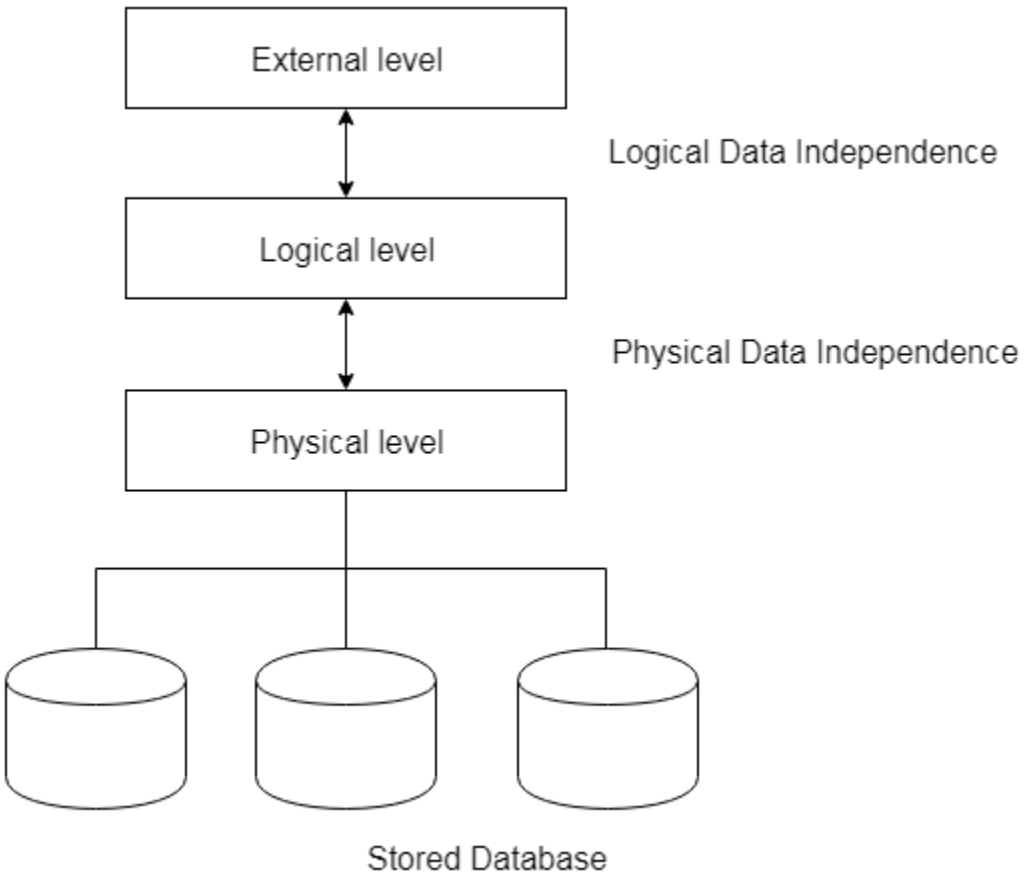
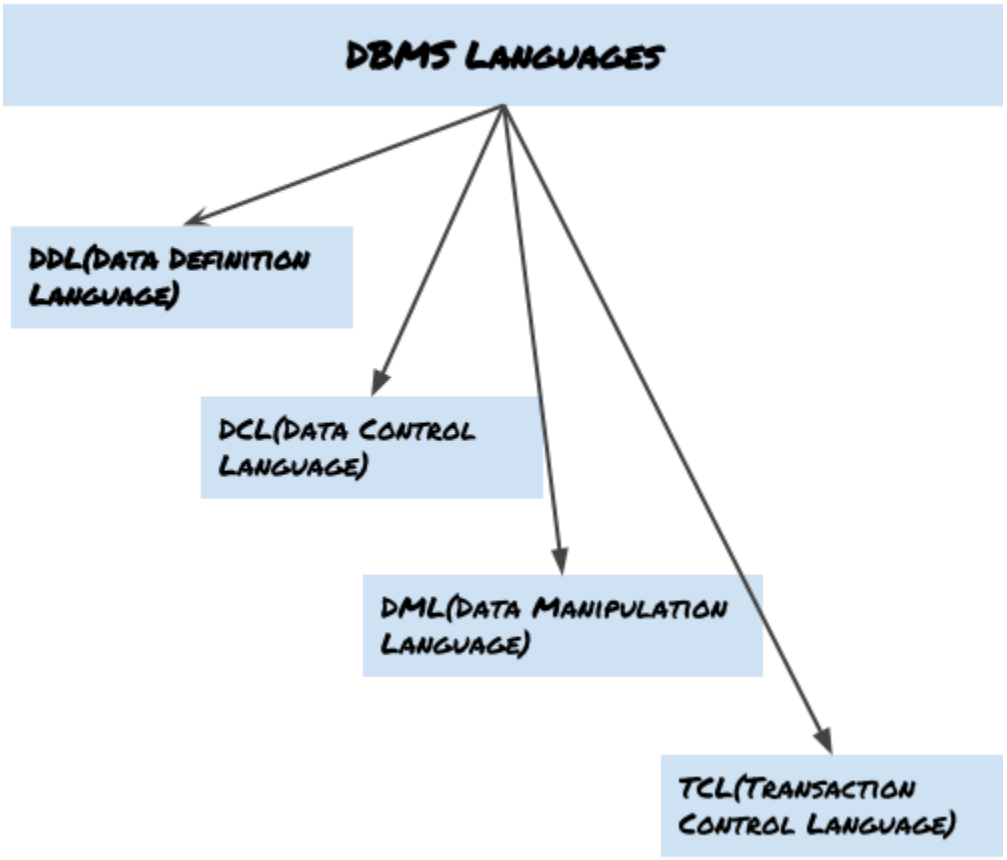


Fig: Data Independence

DBMS languages

Database languages are used to read, update and store data in a database. There are several such languages that can be used for this purpose; one of them is SQL (Structured Query Language).

Types of DBMS languages:



Data Definition Language (DDL)

DDL is used for specifying the database schema. It is used for creating tables, schema, indexes, constraints etc. in database. Lets see the operations that we can perform on database using DDL:

- To create the database instance – CREATE
- To alter the structure of database – **ALTER**
- To drop database instances – DROP
- To delete tables in a database instance – **TRUNCATE**
- To rename database instances – **RENAME**
- To drop objects from database such as tables – **DROP**
- To Comment – **Comment**

All of these commands either defines or update the database schema that's why they come under Data Definition language.

Data Manipulation Language (DML)

DML is used for accessing and manipulating data in a database. The following operations on database comes under DML:

- To read records from table(s) – SELECT
- To insert record(s) into the table(s) – **INSERT**
- Update the data in table(s) – UPDATE
- Delete all the records from the table – DELETE

Data Control language (DCL)

DCL is used for granting and revoking user access on a database –

- To grant access to user – GRANT
- To revoke access from user – REVOKE

In practical data definition language, data manipulation language and data control languages are not separate language, rather they are the parts of a single database language such as SQL.

Transaction Control Language (TCL)

The changes in the database that we made using DML commands are either performed or rollbacked using TCL.

- To persist the changes made by DML commands in database – COMMIT
- To rollback the changes made to the database – ROLLBACK

Interfaces in DBMS

A database management system (DBMS) interface is a user interface which allows for the ability to input queries to a database without using the query language itself.

User-friendly interfaces provide by DBMS may include the following:

1. **Menu-Based Interfaces for Web Clients or Browsing –**
   These interfaces present the user with lists of options (called menus) that lead the user through the formation of a request. Basic advantage of using menus is that they removes the tension of remembering specific commands and syntax of any query language, rather than query is basically composed step by step by collecting or picking options from a menu that is basically shown by the system. Pull-down menus are a very popular technique in *Web based interfaces*. They are also often used in *browsing interface* which allow a user to look through the contents of a database in an exploratory and unstructured manner.

2. **Forms-Based Interfaces** –
   A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert a new data, or they can fill out only certain entries, in which case the DBMS will redeem same type of data for other remaining entries. This type of forms are usually designed or created and programmed for the users that have no expertise in operating system. Many DBMSs have *forms specification languages* which are special languages that help specify such forms.
   Example: SQL* Forms is a form-based language that specifies queries using a form designed in conjunction with the relational database schema.b>
3. **Graphical User Interface –**
   A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUI's utilize both menus and forms. Most GUIs use a pointing device such as mouse, to pick certain part of the displayed schema diagram.

4. **Natural language Interfaces** –
   These interfaces accept request written in English or some other language and attempt to understand them. A Natural language interface has its own schema, which is similar to the database conceptual schema as well as a dictionary of important words.
   The natural language interface refers to the words in its schema as well as to the set of standard words in a dictionary to interpret the request. If the interpretation is successful, the interface generates a high-level query corresponding to the natural language and submits it to the DBMS for processing, otherwise a dialogue is started with the user to clarify any provided condition or request. The main disadvantage with this is that the capabilities of this type of interfaces are not that much advance.

5. **Speech Input and Output –**
   There is an limited use of speech say it for a query or an answer to a question or being a result of a request it is becoming commonplace Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure, and bank account information are allowed speech for input and output to enable ordinary folks to access this information.
   The Speech input is detected using a predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech take place.

6. **Interfaces for DBA –**
   Most database system contains privileged commands that can be used only by the DBA's staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, reorganizing the storage structures of a databases.

Structure of Database Management System

Database Management System (DBMS) is software that allows access to data stored in a database and provides an easy and effective method of –

- Defining the information.
- Storing the information.
- Manipulating the information.
- Protecting the information from system crashes or data theft.

- Differentiating access permissions for different users.

The database system is divided into three components: Query Processor, Storage Manager, and Disk Storage. These are explained as following below.

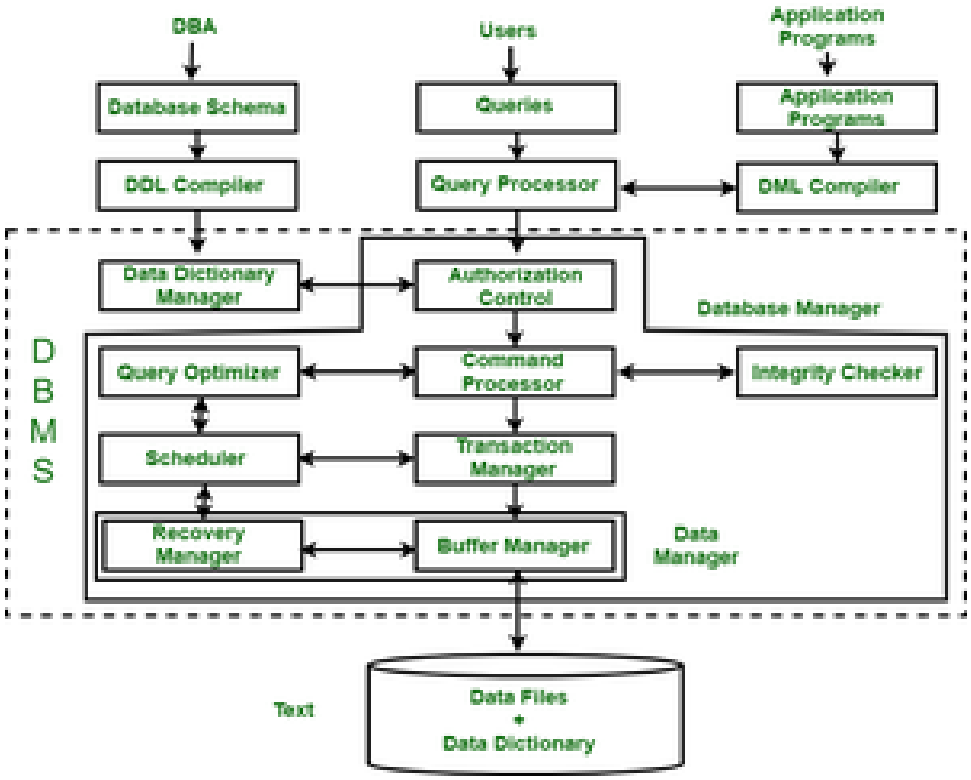


***Figure** – Structure of DBMS*

1. Query Processor:

It interprets the requests (queries) received from end user via an application program into instructions. It also executes the user request which is received from the DML compiler.

Query Processor contains the following components –

- **DML Compiler –**
  It processes the DML statements into low level instruction (machine language), so that they can be executed.

- **DDL Interpreter –**
  It processes the DDL statements into a set of table containing meta data (data about data).

- **Embedded DML Pre-compiler –**
  It processes DML statements embedded in an application program into procedural calls.

- **Query Optimizer –**
  It executes the instruction generated by DML Compiler.

2. Storage Manager:

Storage Manager is a program that provides an interface between the data stored in the database and the queries received. It is also known as Database Control System. It maintains the consistency and integrity of the database by applying the constraints and executes the DCL statements. It is responsible for updating, storing, deleting, and retrieving data in the database.

It contains the following components –

- **Authorization Manager –**
  It ensures role-based access control, i.e,. checks whether the particular person is privileged to perform the requested operation or not.

- **Integrity Manager –**
  It checks the integrity constraints when the database is modified.

- **Transaction Manager –**
  It controls concurrent access by performing the operations in a scheduled way that it receives the transaction. Thus, it ensures that the database remains in the consistent state before and after the execution of a transaction.

- **File Manager –**
  It manages the file space and the data structure used to represent information in the database.

- **Buffer Manager –**
  It is responsible for cache memory and the transfer of data between the secondary storage and main memory.

3. Disk Storage:
It contains the following components –

- **Data Files –**
  It stores the data.

- **Data Dictionary –**
  It contains the information about the structure of any database object. It is the repository of information that governs the metadata.

- **Indices –**
  It provides faster retrieval of data item.

Data Modeling

Data modeling is a technique to document a software system using diagrams and symbols. It is used to represent communication of data.

The highest level of abstraction for the data model is called the Entity Relationship Diagram (ERD). It is a graphical representation of data requirements for a database.

Entity Relationship Diagram

The main value of carefully constructing an ERD is that it can readily be converted into a database structure.

There are three components in ERD.

- Entities: Number of tables you need for your database.
- Attributes: Information such as property, facts you need to describe each table.
- Relationships: How tables are linked together.

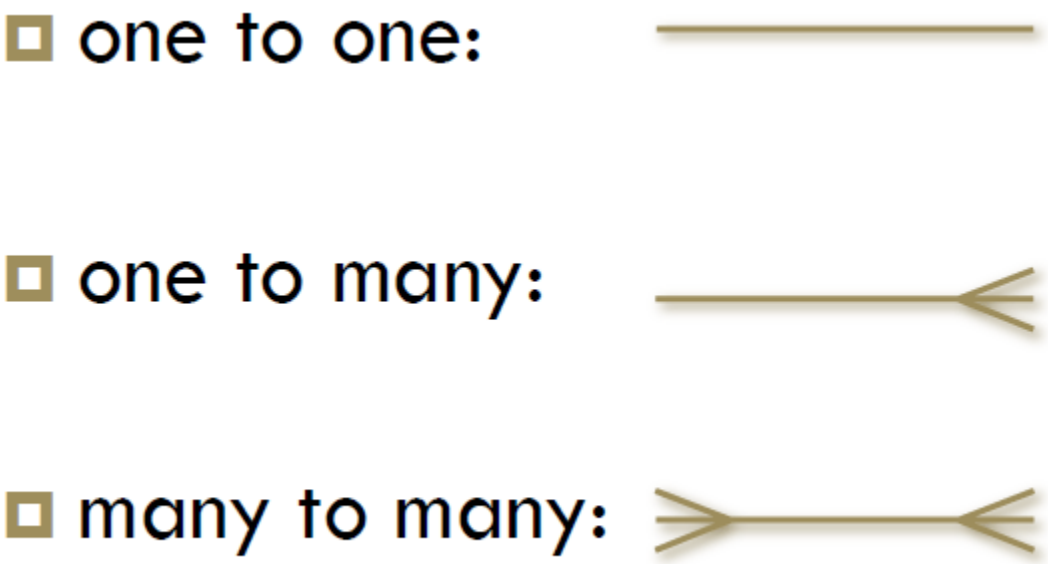
Entity

Entities are the basic objects of ERDs. These are the tables of your database. Entity are nouns and the types usually fall into five classes: concepts, locations, roles, events or things.
For example: students, courses, books, campus, employees, payment, projects.

A specific example of an entity is called an instance. Each instance becomes a record or a row in a table.
For example: the student John Smith is a record in a table called students.

Relationships

Relationships are the associations between the entities. Verbs often describe relationships between entities. We will use Crow's Foot Symbols to represent the relationships. Three types of relationships are discussed in this lab. If you read or hear cardinality ratios, it also refers to types of relationships.



One to One Relationship (1:1)

A single entity instance in one entity class is related to a single entity instance in another entity class.

For example:

- Each student fills one seat and one seat is assigned to only one student.
- Each professor has one office space.

One too Many Relationship (1: M)

A single entity instance in one entity class (parent) is related to multiple entity instances in another entity class (child)

For example:

- One instructor can teach many courses, but one course can only be taught by one instructor.
- One instructor may teach many students in one class, but all the students have one instructor for that class.

Many too Many Relationship (M: M)

Each entity instance in one entity class is related to multiple entity instances in another entity class; and vice versa.

For example:

- Each student can take many classes, and each class can be taken by many students.
- Each consumer can buy many products, and each product can be bought by many consumers.

The detailed Crow's Foot Relationship symbols can be found here. Crow's Foot Relationship Symbols

Many too many relationships are difficult to represent. We need to decompose a many to many (M:M) relationship into two one-to-many (1:M) relationships.

Attributes

Attributes are facts or description of entities. They are also often nouns and become the columns of the table. For example, for entity student, the attributes can be first name, last name, email, address and phone numbers.

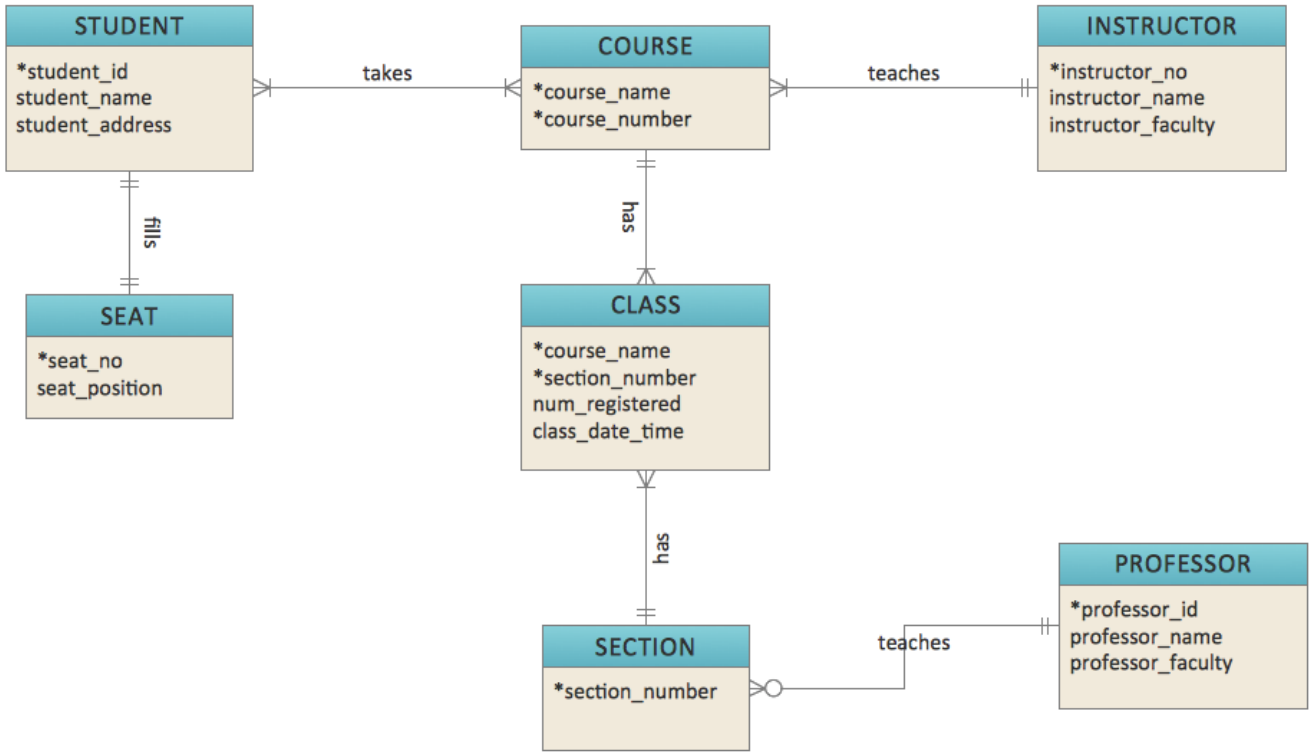
Primary Key

Primary Key* or identifier is an attribute or a set of attributes that uniquely identifies an instance of the entity. For example, for a student entity, student number is the primary key since no two students have the same student number. We can have only one primary key in a table. It identifies uniquely every row and it cannot be null.

Foreign key

A foreign key+ (sometimes called a referencing key) is a key used to link two tables together. Typically you take the primary key field from one table and insert it into the other table where it becomes a foreign key (it remains a primary key in the original table). We can have more than one foreign key in a table.

An Example



ER model

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

For example, Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.

Component of ER Diagram

1. Entity:

An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity

a. Weak Entity

An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.

2. Attribute

The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

For example, id, age, contact number, name, etc. can be attributes of a student.

a. Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined

b. Composite Attribute

An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.

c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

For example, a student can have more than one phone number.

d. Derived Attribute

An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.

3. Relationship

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.

Types of relationship are as follows:

a. One-to-One Relationship

When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

For example, A female can marry to one male, and a male can marry to one female.

b. One-to-many relationship

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.

c. Many-to-one relationship

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

For example, Student enrolls for only one course, but a course can have many students.

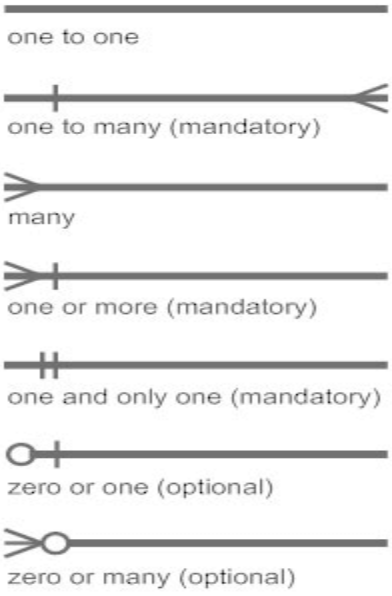
d. Many-to-many relationship

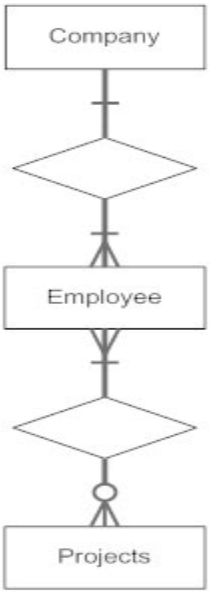
When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.

Notation of ER diagram

Database can be represented using the notations. In ER diagram, many notations are used to express the cardinality. These notations are as follows:

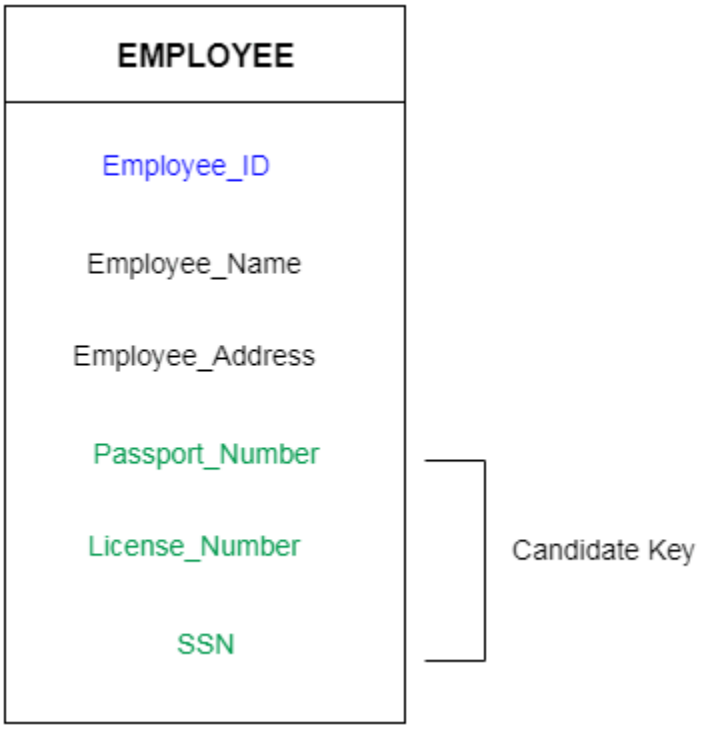




2. Candidate key

- A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
- The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.



3. Super Key

Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.

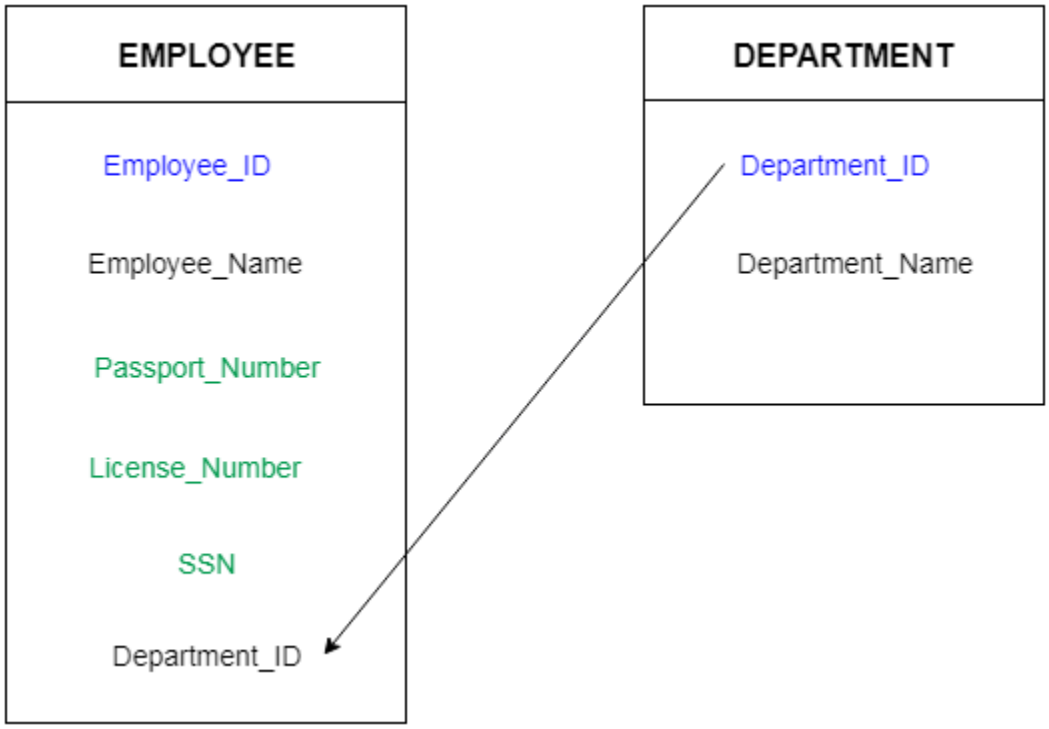
For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.

The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

4. Foreign key

- Foreign keys are the column of the table which is used to point to the primary key of another table.
- In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.

- We add the primary key of the DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



The Enhanced ER Model

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

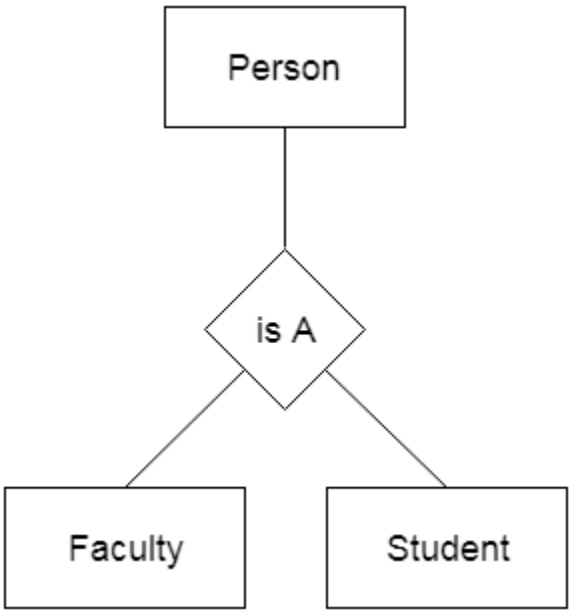
Hence, as part of the **Enhanced ER Model**, along with other improvements, three new concepts were added to the existing ER Model, they were:

1. Generalization
2. Specialization
3. Aggregration

Generalization

- Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.
- In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.
- Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.
- In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass.

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



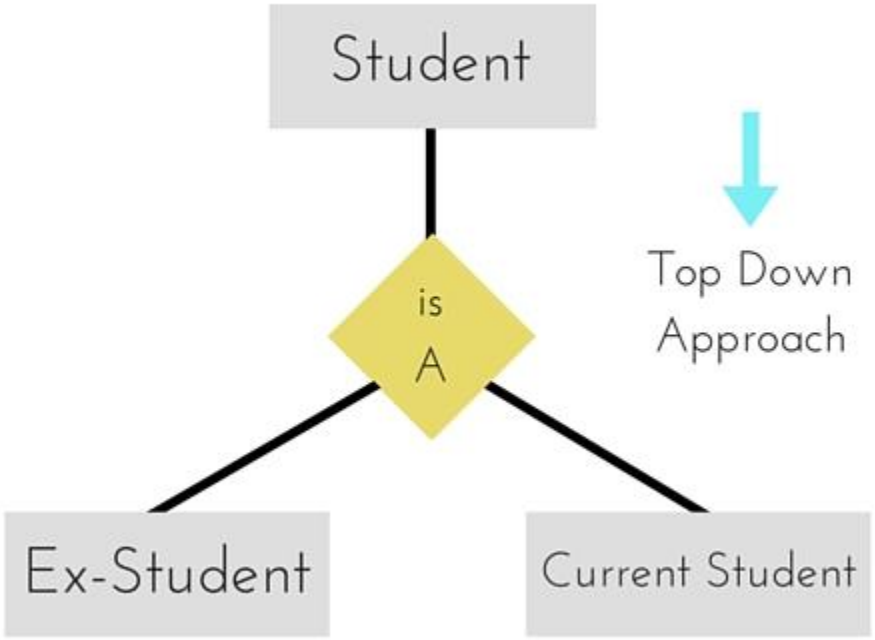
Specialization

Specialization is a top-down approach in which a higher-level entity is divided into multiple *specialized* lower-level entities. In addition to sharing the attributes of the higher-level entity, these lower-level entities have *specific* attributes of their own. Specialization is usually used to find subsets of an entity that has a few different or additional attributes.

The following enhanced entity relationship diagram expresses the entities in a hierarchical database to demonstrate specialization:

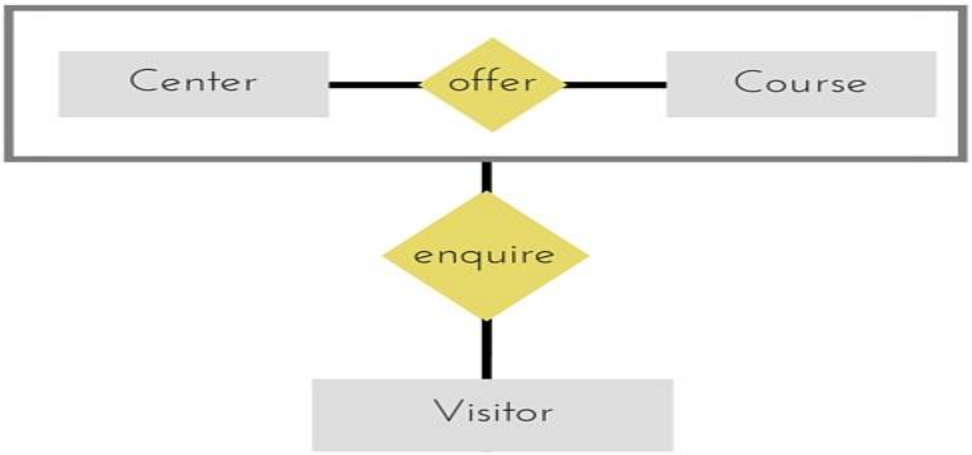
Specialization

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.



Aggregation

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



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

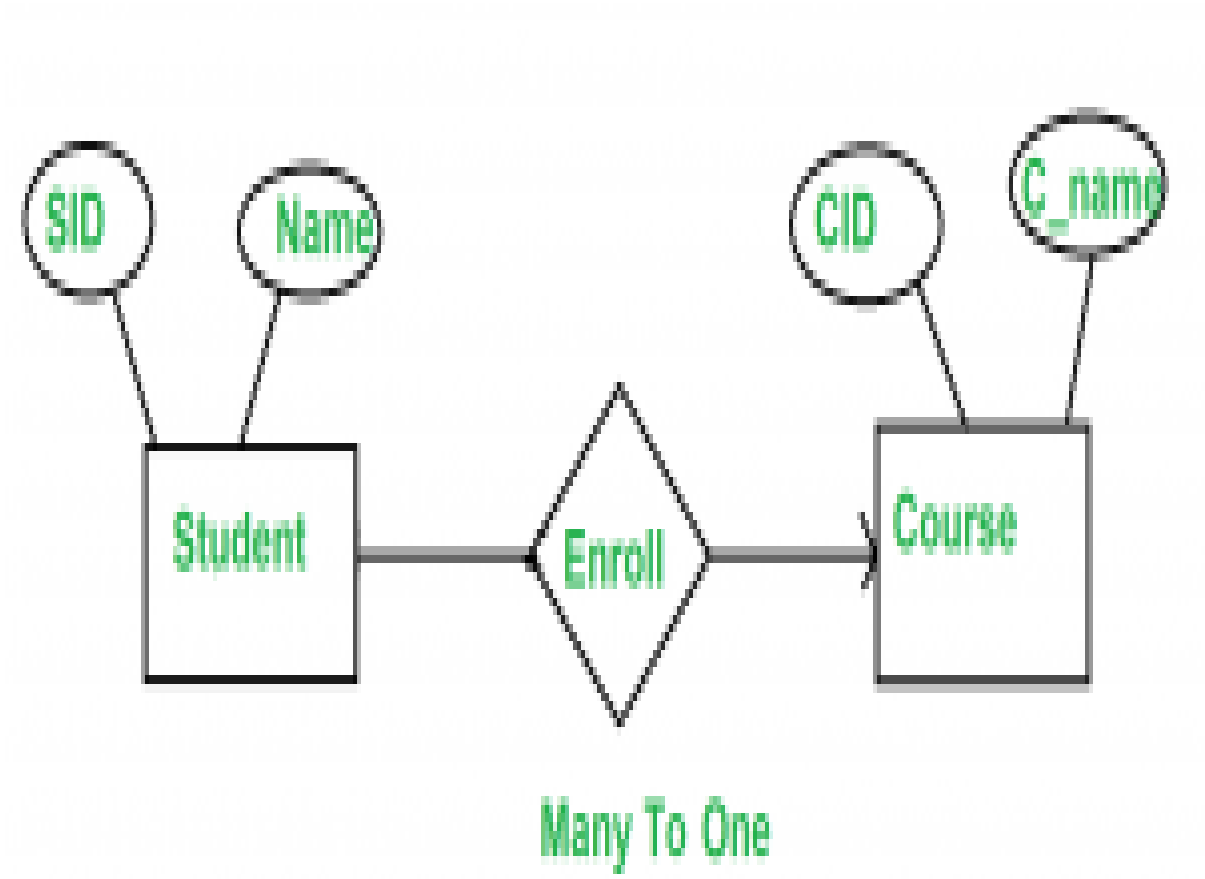
Student visits a Coaching Center, he/she will never enquire about the center only or just about the course, rather he/she will ask enquire about both.

Minimization of ER Diagrams

Entity Relationship (ER) Diagram is diagrammatic representation of data in databases, it shows how data is related.

Note: This article for those who already know what is ER diagram and how to draw ER diagram.

1) When there is One too Many cardinality in ER diagram.
For example, a student can be enrolled only in one course, but a course can be enrolled by many students



For Student(SID, Name), SID is the primary key. For Course (CID, C_name), CID is the primary key

```
      Student                          Course
(SID   Name)                    (CID   C_name)
--------------                  -----------------
  1      A                        c1      Z
  2      B                        c2      Y
  3      C                        c3      X
  4      D
```

```
  Enroll
(SID   CID)
 ----------
 1       C1
 2       C1
 3       c3
 4       C2
```

Now the question is, what should be the primary key for Enroll SID or CID or combined. We can't have CID as primary key as you can see in enroll for the same CID we have multiples SID. (SID , CID) can distinguish table uniquely, but it is not minimum. So SID is the primary key for the relation enroll.

For above ER diagram, we considered three tables in database

```
Student
Enroll
Course
```

But we can combine Student and Enroll table renamed as Student_enroll.

```
 Student_Enroll
( SID   Name   CID )
----------------------
  1      A      c1
  2      B      c1
  3      C      c3
  4      D      c2
```

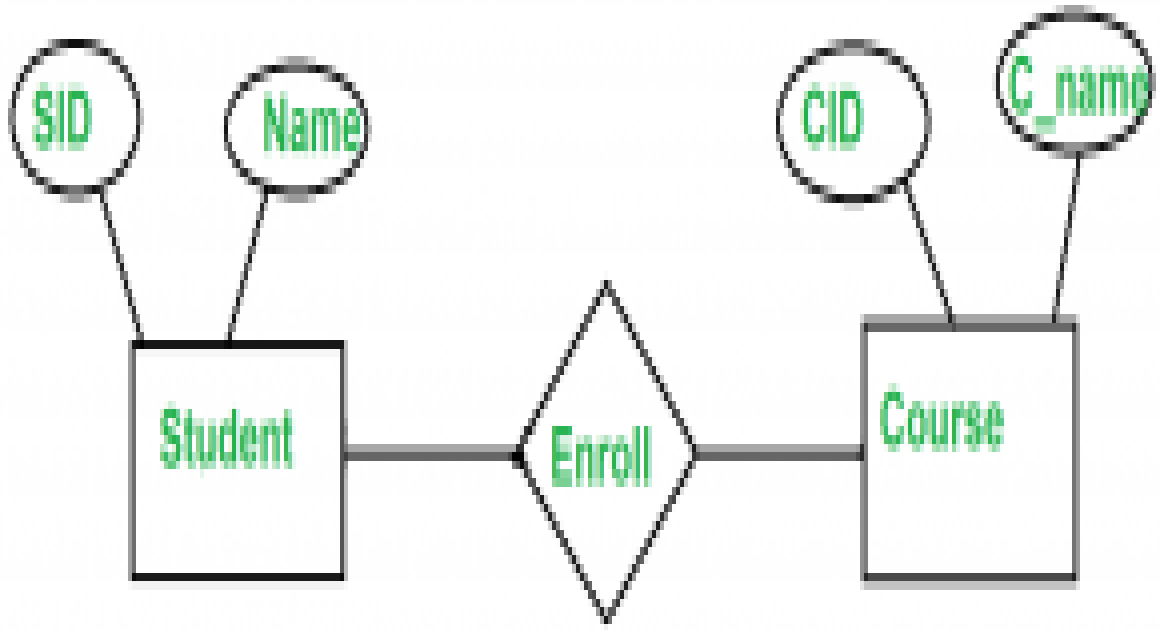
Student and enroll tables are merged now .

So require minimum two DBMS tables for Student_enroll and Course.

Note: In One to Many relationship we can have minimum two tables.

2. When there is Many to Many cardinality in ER Diagram.

Let us consider above example with the change that now student can also enroll more than 1 course.



Many To Many

```
  Student                              Course
( SID   Name)                      ( CID   C_name )
--------------                     -----------------
   1      A                          c1      Z
   2      B                          c2      Y
   3      C                          c3      X
   4      D


              Enroll
           ( SID   CID )
             ----------
             1       C1
```

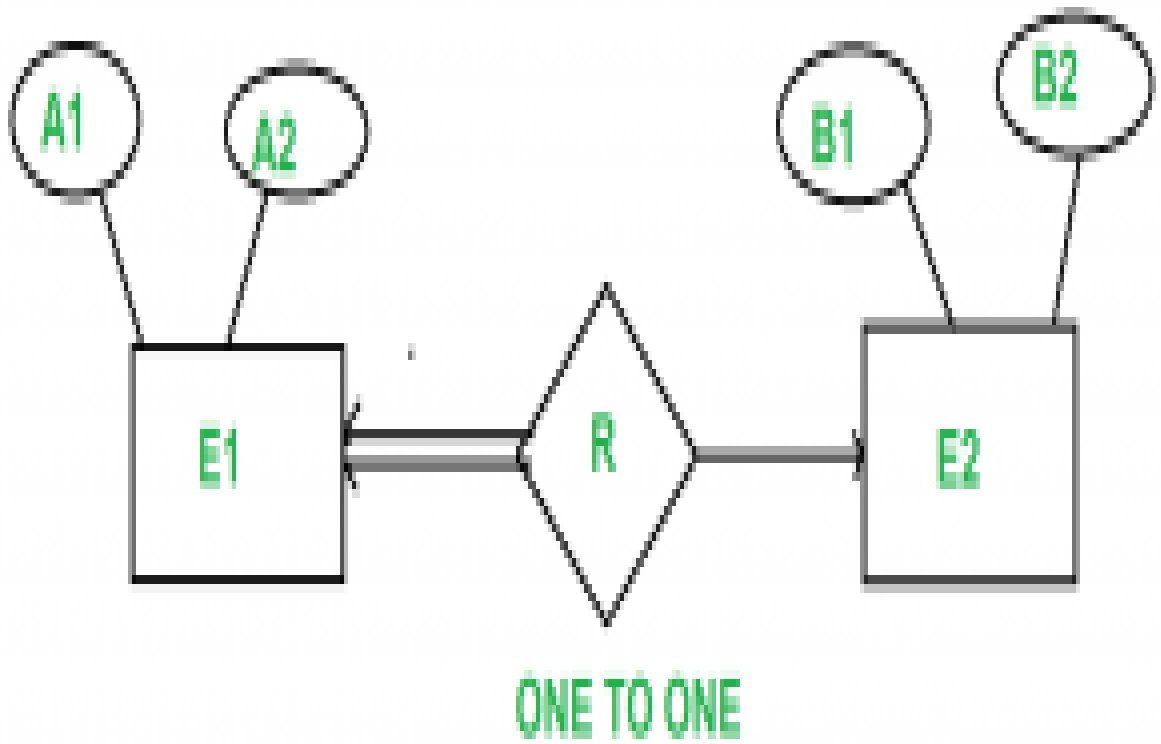
1	C2
2	C1
2	C2
3	c3
4	C2

Now, same question what is the primary key of Enroll relation, if we carefully analyse the Enroll primary key for Enroll
table is (SID , CID).

But in this case we can't merge Enroll table with any one of Student and Course. If we try to merge Enroll with any one of the Student and Course it will create redundant data.

Note: Minimum three tables are required in Many to Many relationship.

3. One to One Relationship
There are two possibilities
A) If we have One to One relationship and we have total participation at at-least one end.
For example, consider the below ER diagram.



A1 and B1 are primary keys of E1 and E2 respectively.

In the above Diagram we have total participation at E1 end.

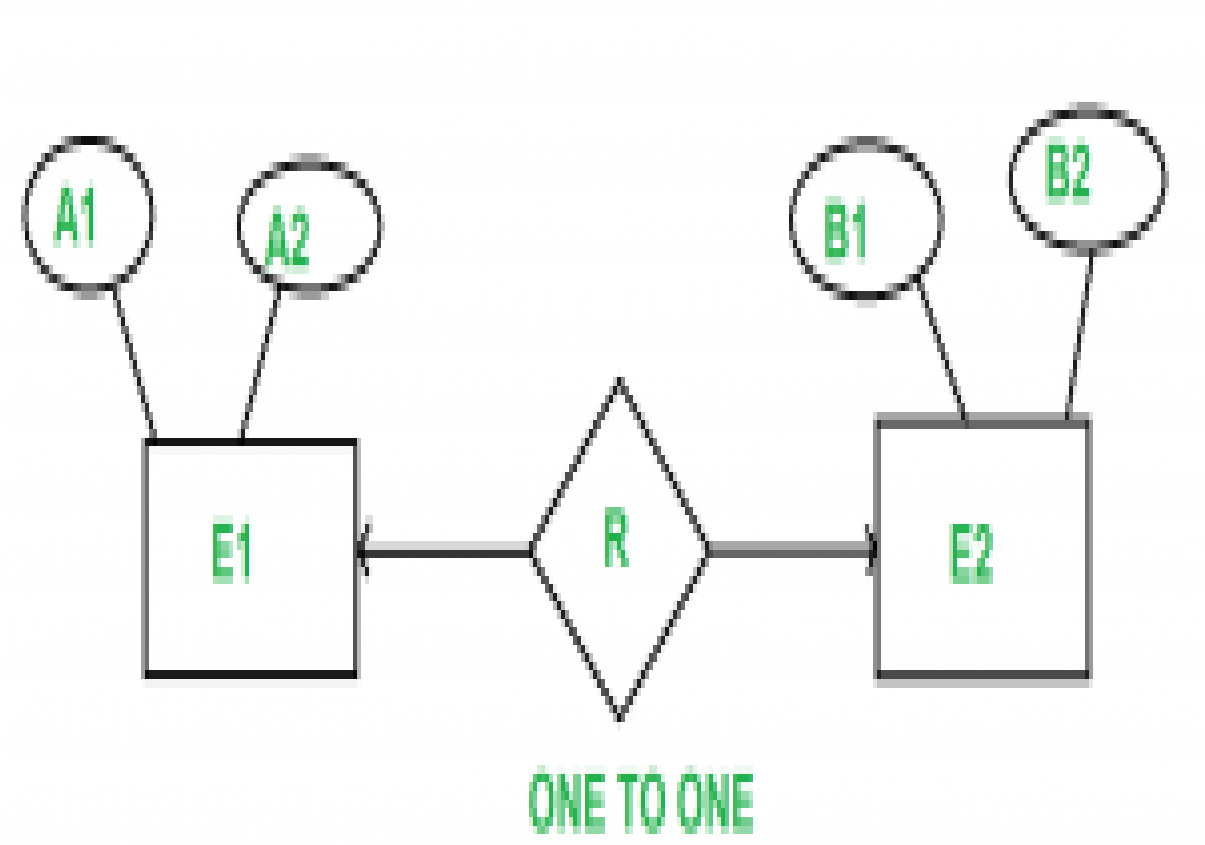
Only a single table is required in this case having primary key of E1 as its primary key.

Since E1 is in total participation, each entry in E1 is related to only one entry in E2, but not all entries in E2 are related to an entry in E1.

The primary key of E1 should be allowed as the primary key of the reduced table, since if the primary key of E2 is used, it might have null values for many of its entries in the reduced table.

Note: Only 1 table required.

B) One to One relationship with no total participation.



A1 and B1 are primary keys of E1 and E2 respectively.

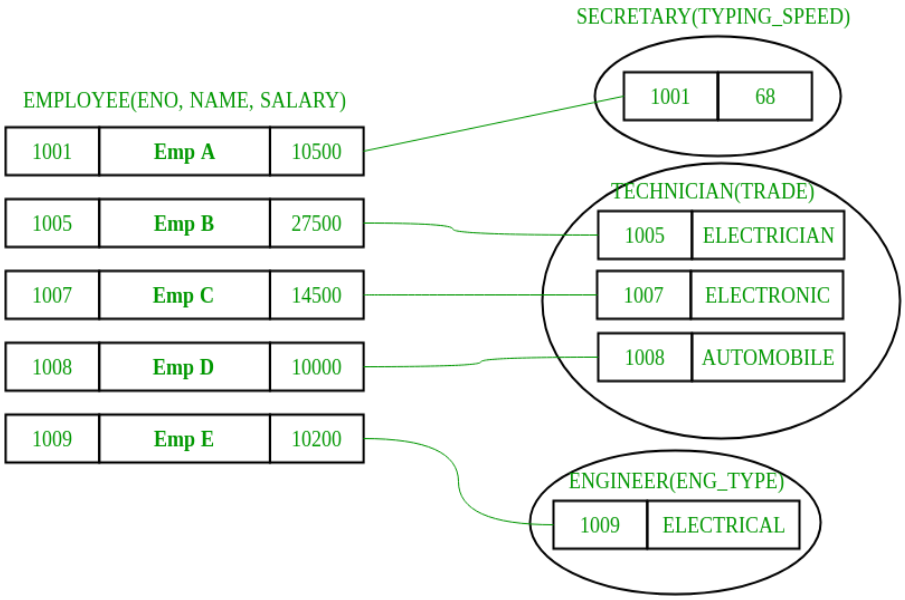
Primary key of R can be A1 or B1, but we can't still combine all the three table into one. if we do, so some entries in combined table may have NULL entries. So idea of merging all three table into one is not good.

But we can merge R into E1 or E2. So minimum 2 tables are required.

Enhanced ER Model

Prerequisite – Introduction of ER Model
Todays time the complexity of the data is increasing so it becomes more and more difficult to use the traditional ER model for database modeling. To reduce this complexity of modeling we have to make improvements or enhancements were made to the existing ER model to make it able to handle the complex application in a better way.
Enhanced entity-relationship diagrams are advanced database diagrams very similar to regular ER diagrams which represents requirements and complexities of complex databases.
It is a diagrammatic technique for displaying the Sub Class and Super Class; Specialization and Generalization; Union or Category; Aggregation etc.
Generalization and Specialization –
These are very common relationship found in real entities. However this kind of relationships was added later as enhanced extension to classical ER model. **Specialized class** are often called as **subclass** while **generalized class** are called superclass, probably inspired by object oriented programming. A sub-class is best understood by **"IS-A analysis"**. Following statements hopefully makes some sense to your mind "Technician IS-A Employee", "Laptop IS-A Computer".
An entity is specialized type/class of other entity. For example, Technician is special Employee in a university system Faculty is special class of Employee. We call this phenomenon as generalization/specialization. In the example here Employee is generalized entity class while Technician and Faculty are specialized class of Employee.

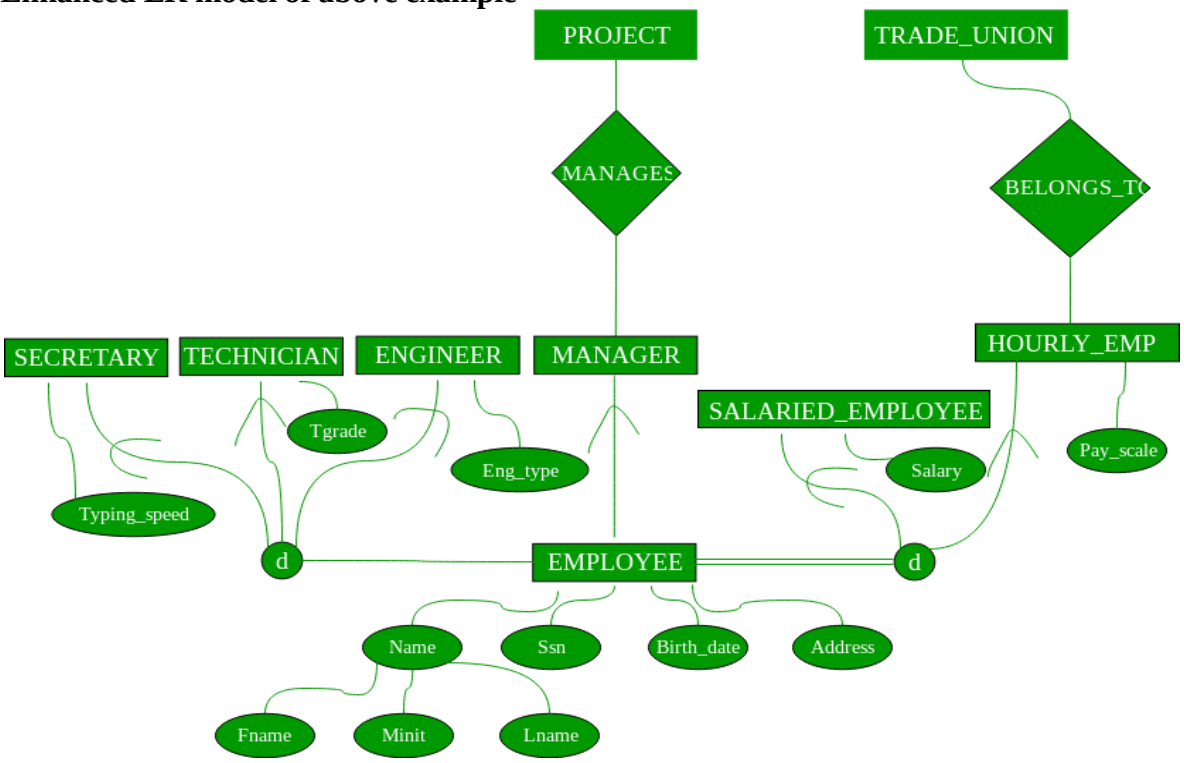
Example – This example instance of **"sub-class"** relationships. Here we have four sets employee: Secretary, Technician, and Engineer. Employee is super-class of rest three set of individual sub-class is subset of Employee set.



- An entity belonging to a sub-class is related with some super-class entity. For instance emp no 1001 is a secretary, and his typing speed is 68. Emp no 1009 is engineer (sub-class) and her trade is "Electrical", so forth.

- Sub-class entity “inherits” all attributes of super-class; for example employee 1001 will have attributes eno, name, salary, and typing speed.

Enhanced ER model of above example –



Constraints – There are two types of constraints on “Sub-class” relationship.

1. **Total or Partial –** A sub-classing relationship is total if every super-class entity is to be associated with some sub-class entity, otherwise partial. Sub-class “job type based employee category” is partial sub-classing – not necessary every employee is one of (secretary, engineer, and technician), i.e. union of these three types is proper subset of all employees. Whereas other sub-classing “Salaried Employee AND Hourly Employee” is total; union of entities from sub-classes is equal to total employee set, i.e. every employee necessarily has to be one of them.
2. **Overlapped or Disjoint –** If an entity from super-set can be related (can occur) in multiple sub-class sets, then it is overlapped sub-classing, otherwise disjoint. Both the examples: job-type based and salaries/hourly employee sub-classing are disjoint.

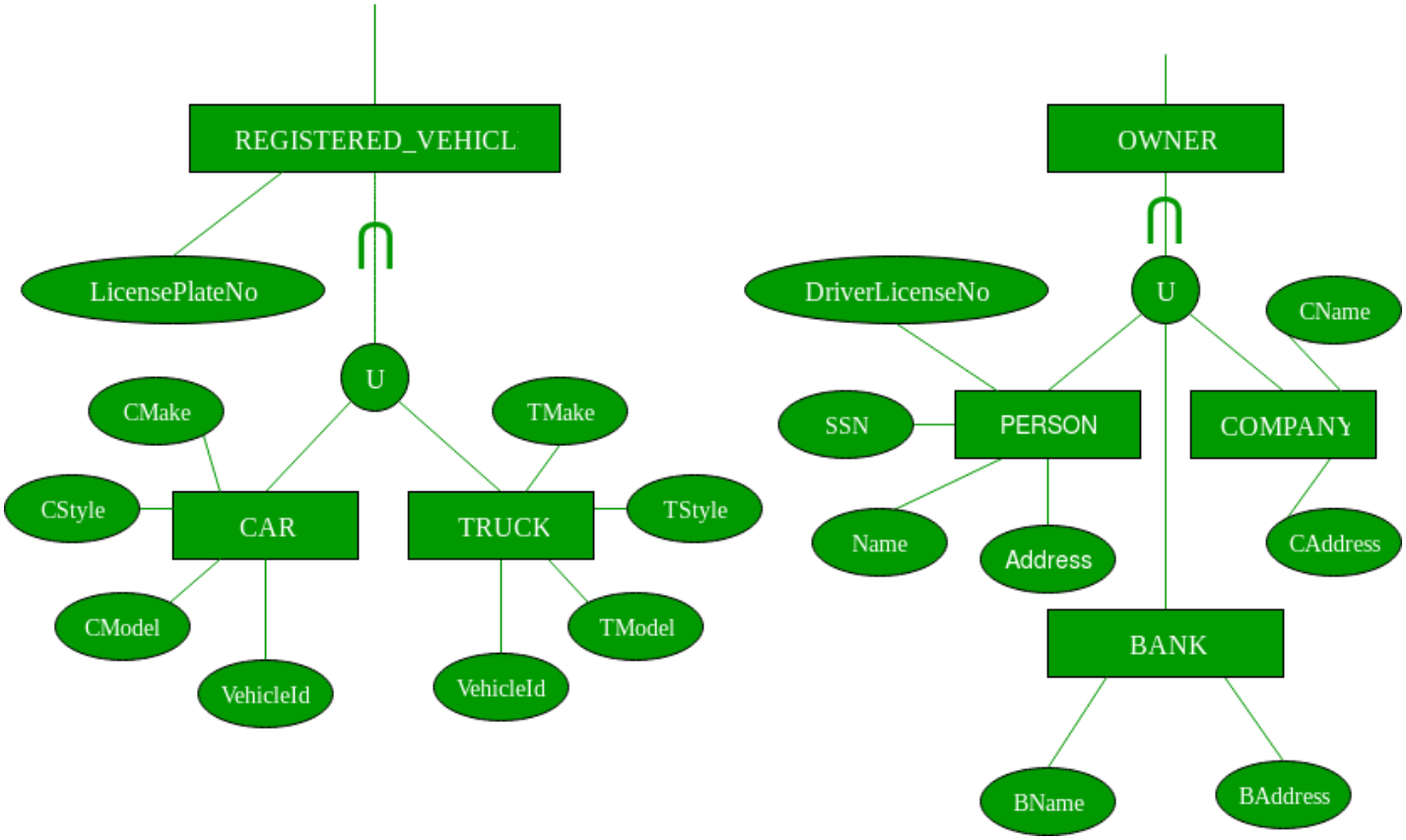
Note – These constraints are independent of each other: can be “overlapped and total or partial” or “disjoint and total or partial”. Also sub-classing has transitive property.

Multiple Inheritance (sub-class of multiple super classes) –

An entity can be sub-class of multiple entity types; such entities are sub-class of multiple entities and have multiple super-classes; Teaching Assistant can subclass of Employee and Student both. A faculty in a university system can be sub-class of Employee and Alumnus both. In multiple inheritance, attributes of sub-class is union of attributes of all super-classes.

Union –

- Set of Libray Members is **UNION** of Faculty, Student, and Staff. A union relationship indicates either of type; for example: a library member is either Faculty or Staff or Student.
- Below are two examples shows how **UNION** can be depicted in ERD – Vehicle Owner is UNION of PERSON and Company, and RTO Registered Vehicle is UNION of Car and Truck.



You might see some confusion in Sub-class and UNION; consider example in above figure Vehicle is super-class of CAR and Truck; this is very much the correct example of subclass as well but here use it different we are saying RTO Registered vehicle is UNION of Car and Vehicle, they do not inherit any attribute of Vehicle, attributes of car and truck are altogether independent set, where is in sub-classing situation car and truck would be inheriting the attribute of vehicle class. Below is Vehicle as modeled as class of Car and Truck.

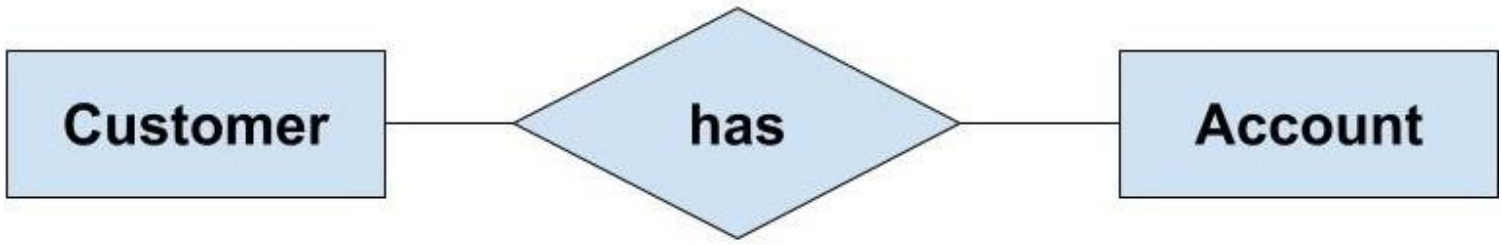
UNIT-2

The degree of relation in DBMS

Degree of Relationship

The degree of a relationship is the number of entity types that participate(associate) in a relationship. By seeing an E-R diagram, we can simply tell the degree of a relationship i.e **the number of an entity type that is connected to a relationship is the degree of that relationship**.

For example, If we have two entity type 'Customer' and 'Account' and they are linked using the primary key and foreign key. We can say that the degree of relationship is 2 because here two entities are taking part in the relationship.



degree 2

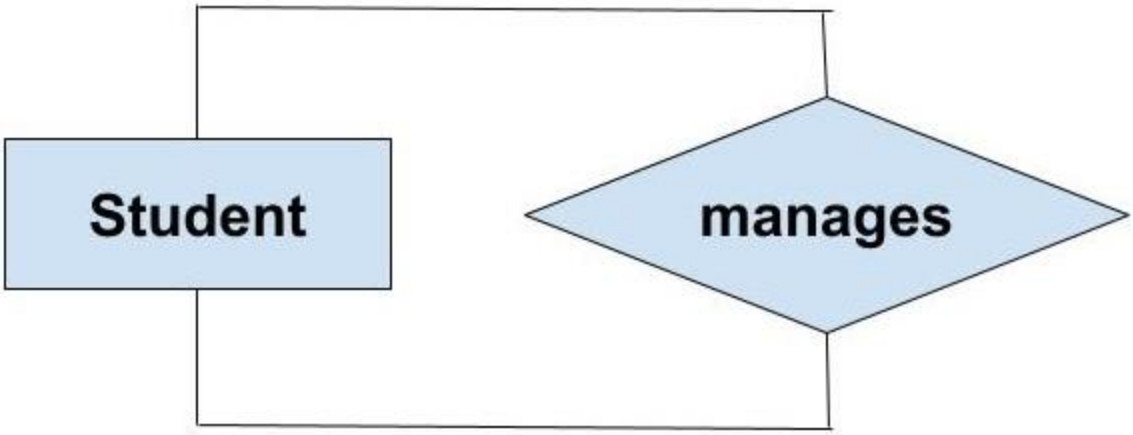
Based on the number of entity types that are connected we have the following degree of relationships:

- Unary
- Binary
- Ternary
- N-ary

Unary (degree 1)

A unary relationship exists when both the participating entity type are the same. When such a relationship is present we say that the degree of relationship is 1.

For example, Suppose in a classroom, we have many students who belong to a particular club-like dance club, basketball club etc. and some of them are club leads. So, a particular group of student is managed by their respective club lead. Here, the group is formed from students and also, the club leads are chosen from students. So, the 'Student' is the only entity participating here. We can represent this relationship using the E-R diagram as follows:



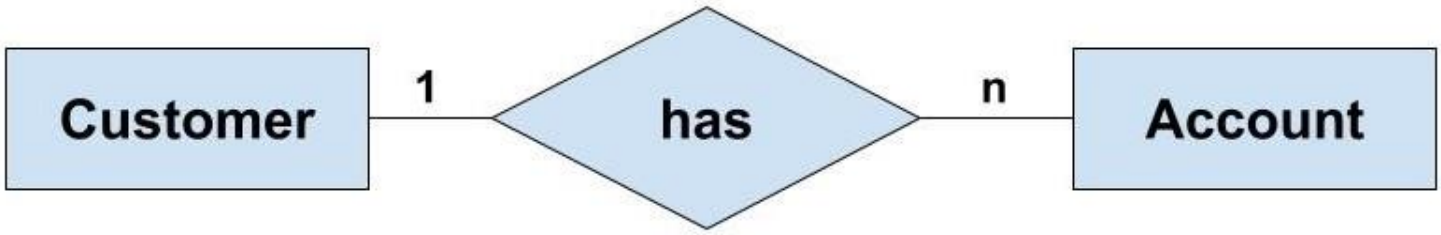
Unary Relationship
(degree 1)

So, here we get the answer to our first question which was asked in the introduction section. Yes, there can be only one entity type in a relationship and the minimum degree of a relationship can be one.

Binary (degree 2)

A binary relationship exists when exactly two entity type participates. When such a relationship is present we say that the degree is 2. This is the most common degree of relationship. It is easy to deal with such relationship as these can be easily converted into relational tables.

For example, we have two entity type 'Customer' and 'Account' where each 'Customer' has an 'Account' which stores the account details of the 'Customer'. Since we have two entity types participating we call it a binary relationship. Also, one 'Customer' can have many 'Account' but each 'Account' should belong to only one 'Customer'. We can say that it is a one-to-many binary relationship.

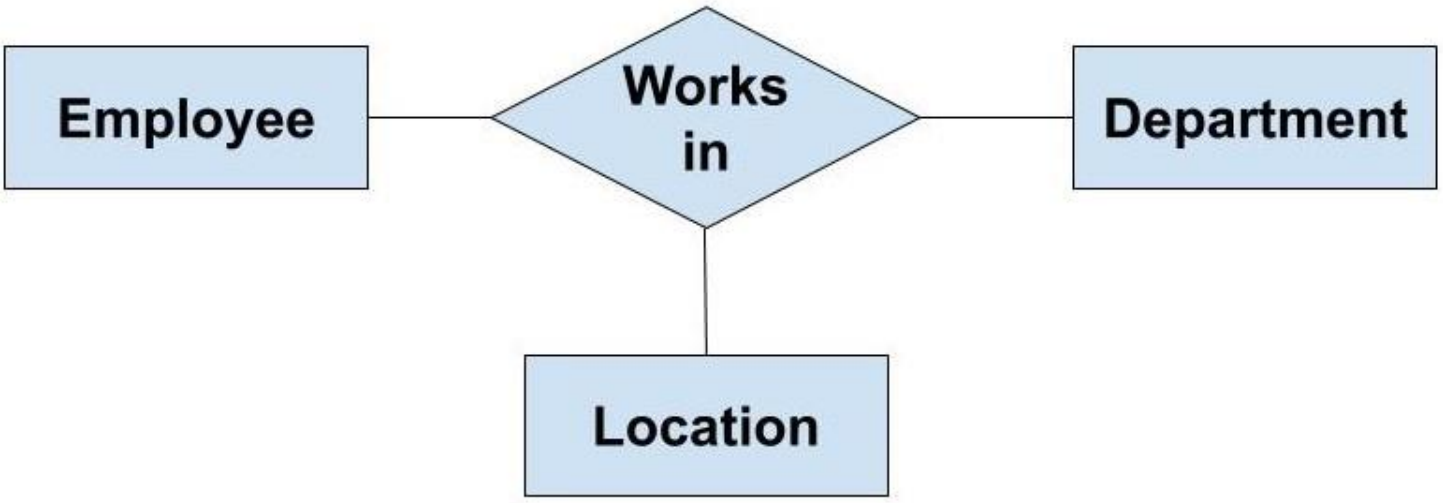


Binary Relationship
(degree 2)

Ternary (degree 3)

A ternary relationship exists when exactly three entity type participates. When such a relationship is present we say that the degree is 3. As the number of entity increases in the relationship, it becomes complex to convert them into relational tables.

For example, we have three entity type 'Employee', 'Department' and 'Location'. The relationship between these entities is defined as an employee works in a department, an employee works at a particular location. So, we can see we have three entities participating in a relationship so it is a ternary relationship. The degree of this relation is 3.

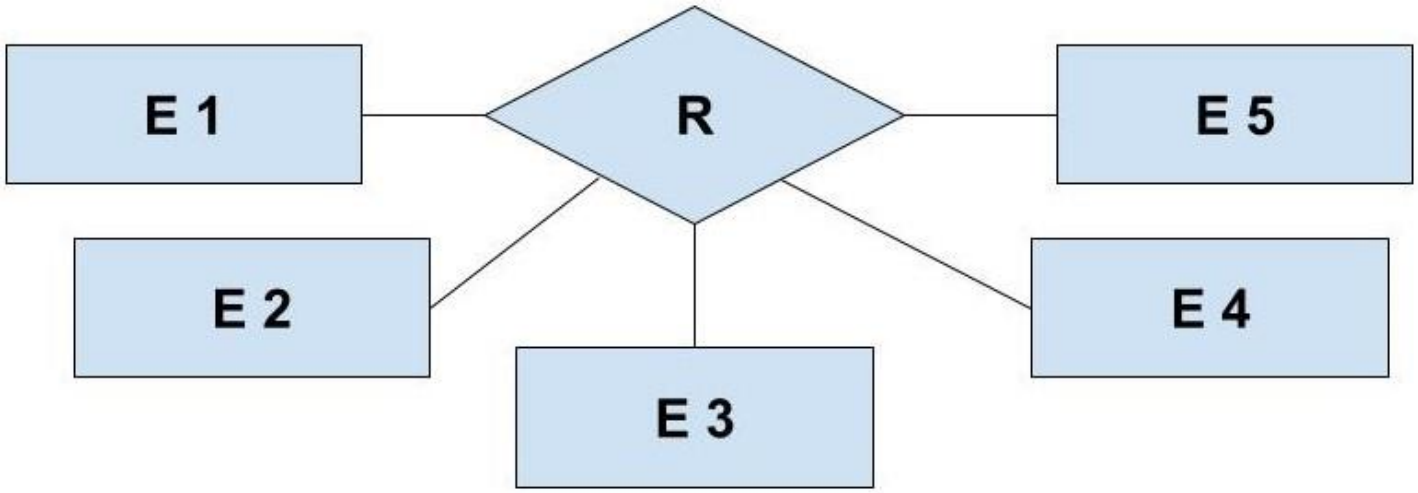


Ternary Relationship
(degree 3)

N-ary (n degree)

An N-ary relationship exists when ‘n’ number of entities is participating. So, any number of entities can participate in a relationship. There is no limitation to the maximum number of entities that can participate. But, relations with a higher degree are not common. This is because the conversion of higher degree relations to relational tables gets complex. We are making an E-R model because it can be easily be converted into any other model for implementing the database. But, this benefit is not available if we use higher degree relations. So, binary relations are more popular and widely used. Though we can make a relationship with any number of entity types but we don't do that.

We represent an N-ary relationship as follows:



N-ary Relationship

In the above example, E1 denotes the first entity type, E2 denotes the second entity type and so on. R represents the relationship. So, here we have a total of 5 entity type which participates in the relationship. Therefore, the degree of the above n-ary relationship is 5.

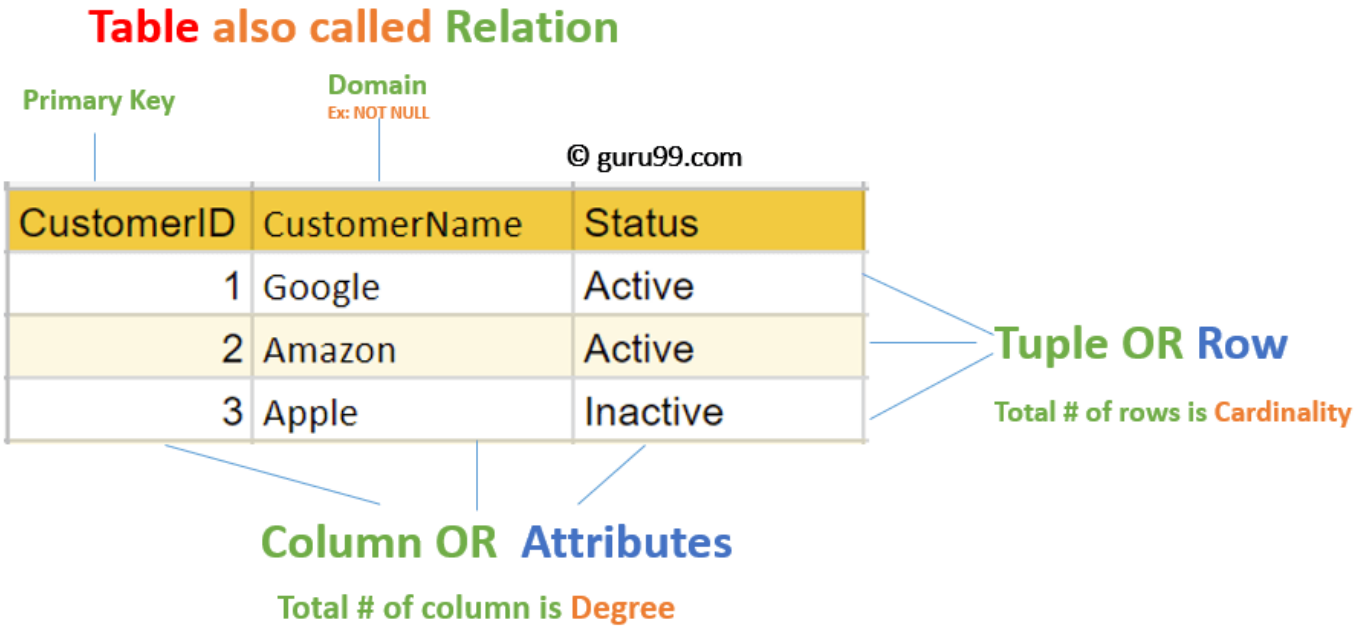
What is Relational Model?

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Relational Model Concepts

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.
2. **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3. **Tuple** – It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.
8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key** - Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain



Relational Integrity Constraints

Relational Integrity constraints in DBMS are referred to conditions which must be present for a valid relation. These Relational constraints in DBMS are derived from the rules in the mini-world that the database represents.

There are many types of Integrity Constraints in DBMS. Constraints on the Relational database management system is mostly divided into three main categories are:

1. Domain Constraints
2. Key Constraints
3. Referential Integrity Constraints

Domain Constraints

Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Example:

```
Create DOMAIN CustomerName
CHECK (value not NULL)
```

The example shown demonstrates creating a domain constraint such that CustomerName is not NULL

Key Constraints

An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

Example:

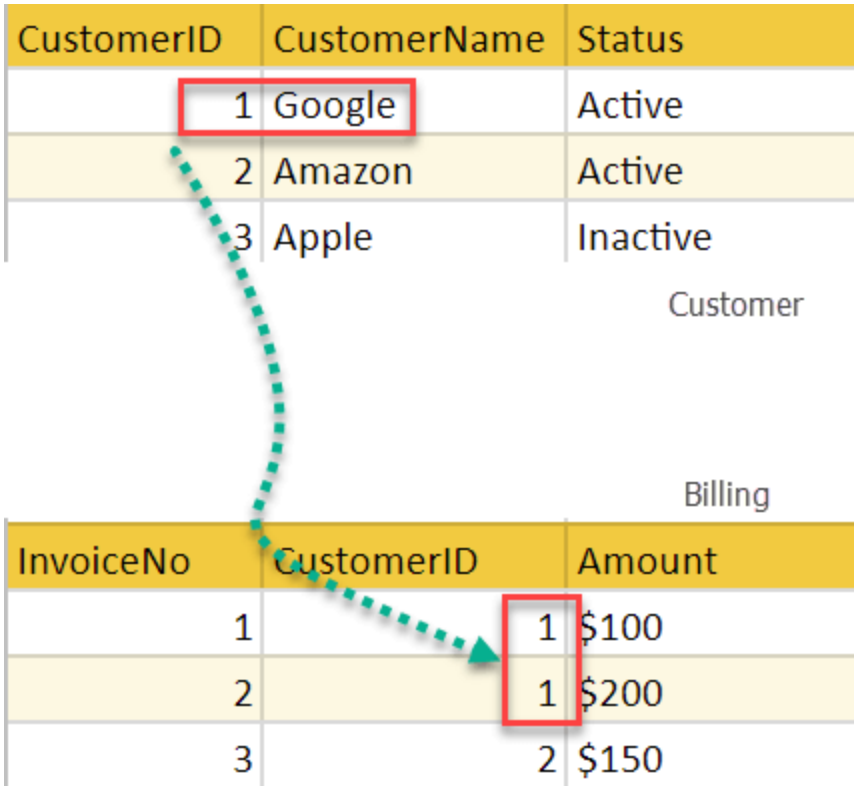
In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Referential Integrity Constraints

Referential Integrity constraints in DBMS are based on the concept of Foreign Keys. A foreign key is an important attribute of a relation which should be referred to in other relationships. Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation. However, that key element must exist in the table.

Example:



In the above example, we have 2 relations, Customer and Billing.

Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount $300

Operations in Relational Model

Four basic update operations performed on relational database model are

Insert, update, delete and select.

- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

Insert Operation

The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Update Operation

You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

Delete Operation

To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

In the above-given example, CustomerName= "Apple" is deleted from the table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same database.

Select Operation

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

CustomerID	CustomerName	Status
2	Amazon	Active

In the above-given example, CustomerName="Amazon" is selected

Best Practices for creating a Relational Model

- Data need to be represented as a collection of relations
- Each relation should be depicted clearly in the table
- Rows should contain data about instances of an entity
- Columns must contain data about attributes of the entity
- Cells of the table should hold a single value
- Each column should be given a unique name
- No two rows can be identical
- The values of an attribute should be from the same domain

Advantages of using Relational Model

- **Simplicity**: A Relational data model in DBMS is simpler than the hierarchical and network model.
- **Structural Independence**: The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
- **Easy to use**: The Relational model in DBMS is easy as tables consisting of rows and columns are quite natural and simple to understand
- **Query capability**: It makes possible for a high-level query language like SQL to avoid complex database navigation.
- **Data independence**: The Structure of Relational database can be changed without having to change any application.
- **Scalable**: Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

Disadvantages of using Relational Model

- Few relational databases have limits on field lengths which can't be exceeded.
- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

Relational Algebra

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

Types of Relational operation

1. Select Operation:

- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).

1. Notation: σ p(r)

Where:

σ is used for selection prediction
r is used for relation
p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input:

1. σ BRANCH_NAME="perryride" (LOAN)

Output:

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

2. Project Operation:

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by ∏.

1. Notation: ∏ A1, A2, A3 (r)

Where

A1, **A2**, **A3** is used as an attribute name of relation **r**.

Example: CUSTOMER RELATION

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input:

1. ∏ NAME, CITY (CUSTOMER)

Output:

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

3. Union Operation:

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by ∪.

1. Notation: R ∪ S

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11

Williams	L-17

Input:

1. ∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

4. Set Intersection:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection ∩.

1. Notation: R ∩ S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Smith
Jones

5. Set Difference:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).

1. Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Jackson
Hayes

Willians	
Curry	

6. Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.

I. Notation: E X D

Example:

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Input:

1. EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO
1	Smith	A	A
1	Smith	A	B
1	Smith	A	C
2	Harry	C	A
2	Harry	C	B
2	Harry	C	C
3	John	B	A
3	John	B	B
3	John	B	C

7. Rename Operation:

The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

1. ρ(STUDENT1, STUDENT)

Note: Apart from these common operations Relational algebra can be used in Join operations.

Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Example:

EMPLOYEE

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

SALARY

EMP_CODE	SALARY
101	50000
102	30000
103	25000

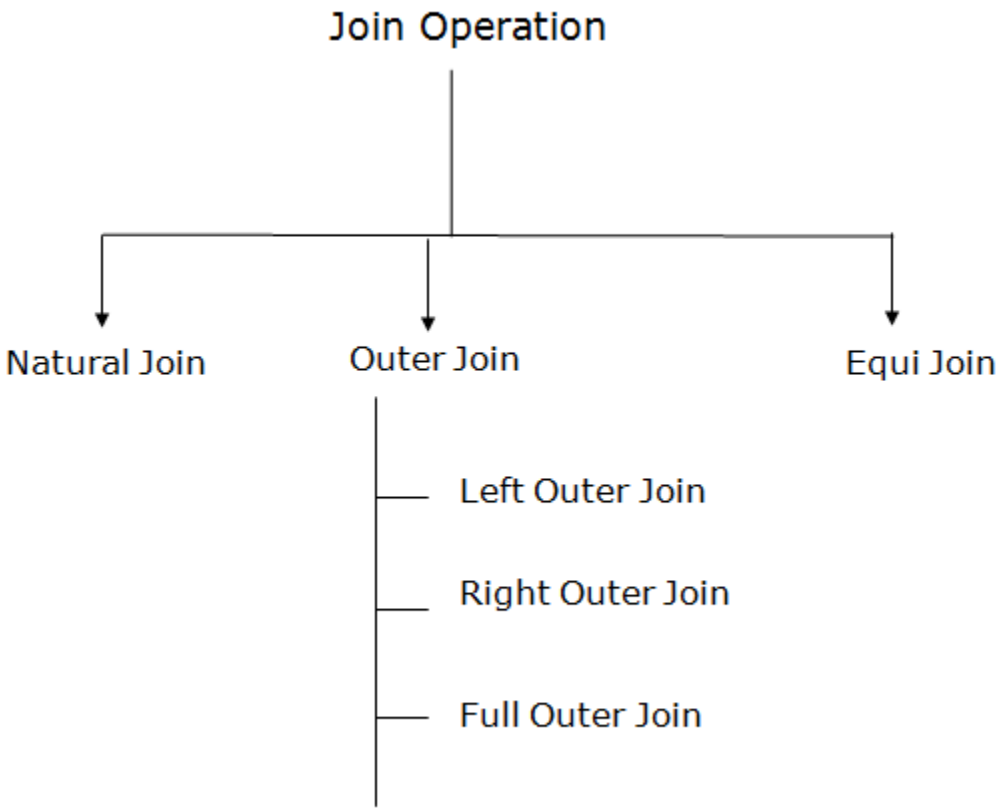
1. Operation: (EMPLOYEE ⋈ SALARY)

Result:

EMP_CODE	EMP_NAME	SALARY
101	Stephan	50000

102	Jack	30000
103	Harry	25000

Types of Join operations:



1. Natural Join:

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- It is denoted by ⋈.

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

1. ∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

2. Outer Join:

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

Example:

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000

Kuber	HCL	30000
Hari	TCS	50000

Input:

1. (EMPLOYEE ⋈ FACT_WORKERS)

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

An outer join is basically of three types:

a. Left outer join
b. Right outer join
c. Full outer join

a. Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by ⟕.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⟕ FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY

Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

b. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by ⟖.

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

Input:

1. EMPLOYEE ⟖ FACT_WORKERS

Output:

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

c. Full outer join:

- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- It is denoted by ⟗.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⋈ FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

Input:

1. CUSTOMER ⋈ PRODUCT

Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai

Introduction to Relational Calculus in DBMS

Relational calculus in RDBM is referring to the non-procedural query language that emphasizes on the concept of what to for the data management rather how to do those. The relational calculus provides descriptive information about the queries to achieve the required result by using mathematical predicates calculus notations. It is an integral part of the relational data model. The relational calculus in DBMS uses specific terms such as tuple and domain to describe the queries. Some of the other related common terminologies for

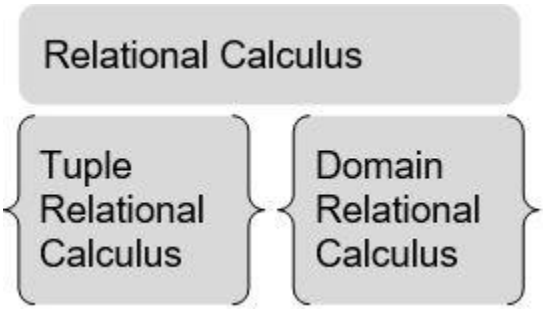
relational calculus are variables, constant, Comparison operators, logical connectives, and quantifiers. It creates the expressions that are also known as formulas with unbound formal variables.

Types of Relational Calculus in DBMS

In this section, we will discuss the types of relational calculus in DBMS based on the terms and process of the mathematical description of queries functionalities. Tuple and domain are the major components of relational calculus. A result tuple is an assignment of constants to these

Variables that make the formula evaluate to be true. There are two types of relational calculus available in DBMS

- Tuple relational calculus (TRC)
- Domain relational calculus (DRC)



Both the types of relational calculus are semantically similar for operating in DBMS data retrieval definitions. We will discuss each type of relational calculus with some database table examples to represent the syntax and its uses.

TRC

Tuple relational calculus works on filtering the tuples based on the specified conditions.TRC is the variable range over the tuples and is a type of simple subset of the first-order logic.TRC considers tuples as equal status as variables, and field referencing can be used to select the tuple parts. It is represented using letter ‘T’ and conditions with the pipe symbol and enclosing curly braces.

Syntax of TRC:

```
{T | Conditions)
```

The TRC syntax supports to denote the Table names or relation names, defining the tuple variables, and the column names. It uses the ‘.’ operator symbol to specify the column names with the table name.

TRC specifies the relation names with the Tuple variable name such as ’T’. Syntax of Relation definition in TRC:

```
Relation(T)
```

For example, if the Product is the relation name, it can be denoted as Product(T). Similarly, TRC has the provision to specify the conditions. The condition is applicable for a particular attribute or the column.

For instance, if the data need to be represented for the particular product id of value 10, it can be denoted as T.product_id=10, where T is the tuple variable that represents the row of the table.

Let us assume the Product table in the database as follows:

Product_id	Product Category	Product Name	Product Unit Price
8	New	TV Unit 1	$100
10	New	TV Unit 2	$120
12	Existing	TV Cabinet	$77

Now to represent the relational calculus to return the product name that has the product id value as 10 from the product table, it can be denoted as with the tuple variable T.

```
T.Product Name | Product(T) AND T.Product_id = 10
```

This relational calculus predicate describes what to do for getting the resultant tuple from the database. The result of the tuple relational calculus for the Product table will be:

Product_id	Product Name
10	TV Unit 2

DRC

The domain regional calculus works based on the filtering of the domain and the related attributes.DRC is the variable range over the domain elements or the filed values. It is a type of simple subset of first-order logic. It is domain-dependent compared to TRC is tuple dependent. In DRC the formal variables are explicit for the relational calculus representations. The domain attributes in DRC can be represented as C1, C2,…, Cn and the condition related to the attributes can be denoted as the formula defining the condition for fetching the F(C1, C2, …Cn)

Syntax of DRC in DBMS

```
{c1, c2,...,cn| F(c1, c2,... ,cn)}
```

Let us assume the same Product table in the database as follows:

Product_id	Product Category	Product Name	Product Unit Price
8	New	TV Unit 1	$100
10	New	TV Unit 2	$120
12	Existing	TV Cabinet	$77

DRC for the product name attribute from the Product table needs where the product id is 10, It will be demoted as:

```
{< Product Name, Product_id> | ∈ Product ∧ Product_id> 10}
```

The result of the domain relational calculus for the Product table will be

Product_id	Product Name
10	TV Unit 2

Some of the commonly used logical operator notations for DRC are ∧ for AND, ∨ for OR, and ¬ for NOT. Similarly, the mathematical symbol ∈ refers to the relation "is an element of" or known as the set membership.

Conclusion

The relational calculus is the schematic description of the queries that provide the structured approach for what the functionalities should be to retrieve and process the data in the relational database. It specifies the scenarios of what to do using the queries that help to implement the syntactical form of the queries in the databases.

UNIT-3

SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

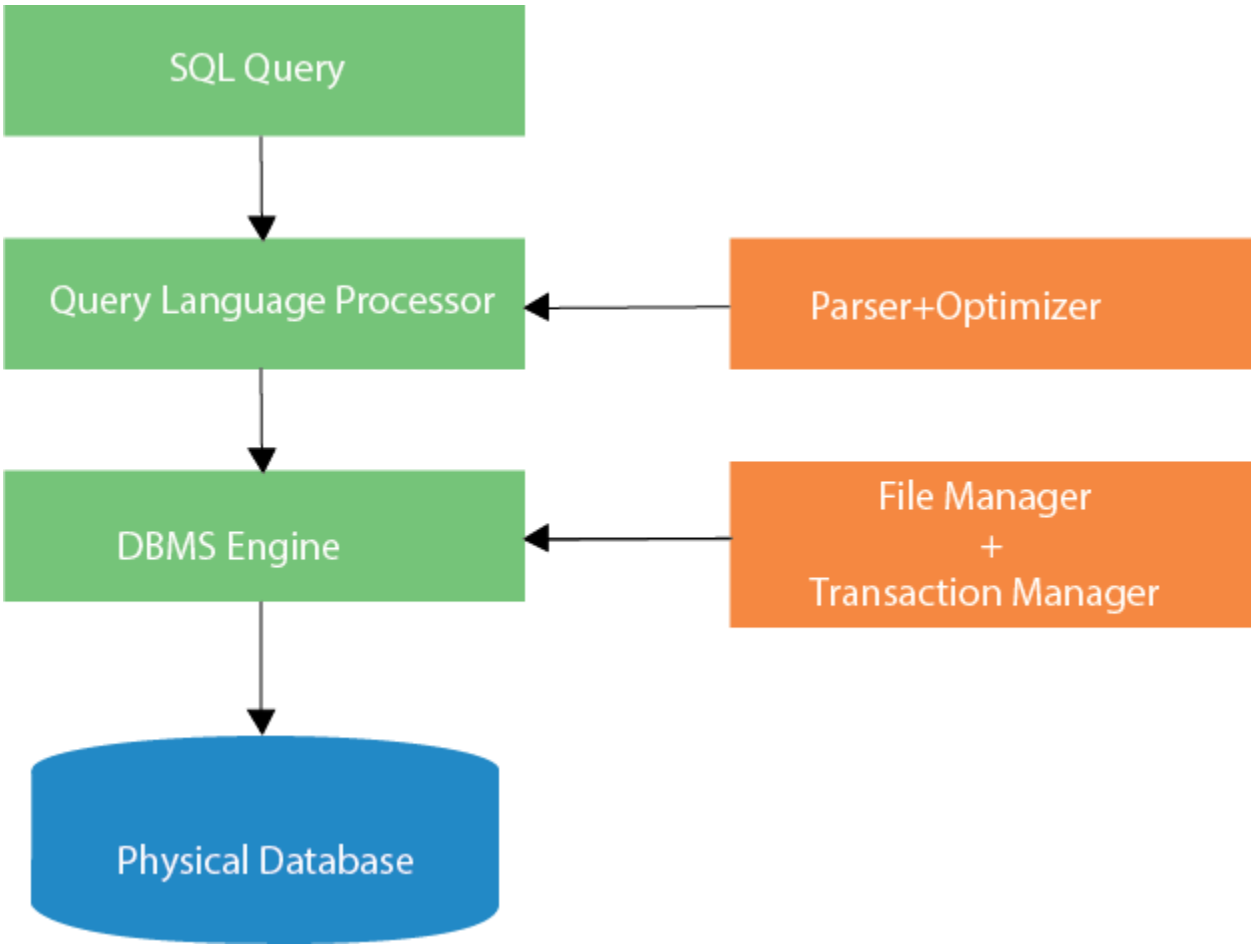
Rules:

SQL follows the following rules:

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

SQL process:

- When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the SQL engine determines that how to interpret the task.
- In the process, various components are included. These components can be optimization Engine, Query engine, Query dispatcher, classic, etc.
- All the non-SQL queries are handled by the classic query engine, but SQL query engine won't handle logical files.



Characteristics of SQL

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create a view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures, and views.

Advantages of SQL

There are the following advantages of SQL:

High speed

Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.

No coding needed

In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

Well defined standards

Long established are used by the SQL databases that are being used by ISO and ANSI.

Portability

SQL can be used in laptop, PCs, server and even some mobile phones.

Interactive language

SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

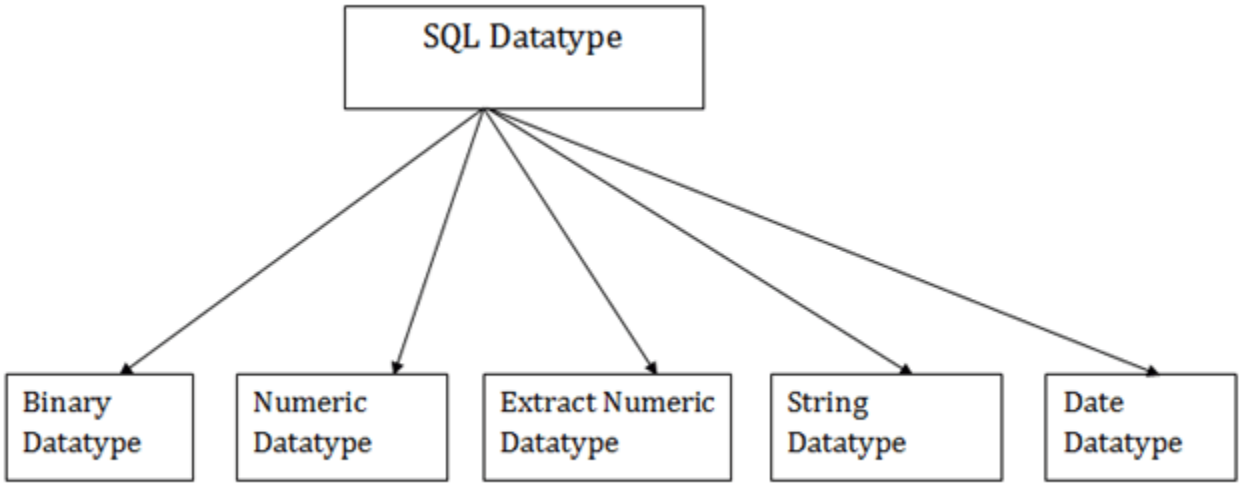
Multiple data view

Using the SQL language, the users can make different views of the database structure.

SQL Datatype

- SQL Datatype is used to define the values that a column can contain.
- Every column is required to have a name and data type in the database table.

Datatype of SQL:



1. Binary Datatypes

There are Three types of binary Datatypes which are given below:

Data Type	Description
binary	It has a maximum length of 8000 bytes. It contains fixed-length binary data.
varbinary	It has a maximum length of 8000 bytes. It contains variable-length binary data.
image	It has a maximum length of 2,147,483,647 bytes. It contains variable-length binary data.

2. Approximate Numeric Data type:

The subtypes are given below:

Data type	From	To	Description
float	-1.79E + 308	1.79E + 308	It is used to specify a floating-point value e.g. 6.2, 2.9 etc.
real	-3.40e + 38	3.40E + 38	It specifies a single precision floating point number

3. Exact Numeric Datatype

The subtypes are given below:

Data type	Description
int	It is used to specify an integer value.
smallint	It is used to specify small integer value.
bit	It has the number of bits to store.
decimal	It specifies a numeric value that can have a decimal number.
numeric	It is used to specify a numeric value.

4. Character String Datatype

The subtypes are given below:

Data type	Description
char	It has a maximum length of 8000 characters. It contains Fixed-length non-unicode characters.

varchar	It has a maximum length of 8000 characters. It contains variable-length non-unicode characters.
text	It has a maximum length of 2,147,483,647 characters. It contains variable-length non-unicode characters.

5. Date and time Datatypes

The subtypes are given below:

Datatype	Description
date	It is used to store the year, month, and days value.
time	It is used to store the hour, minute, and second values.
timestamp	It stores the year, month, day, hour, minute, and the second value.

PL/SQL - Constants and Literals

In this chapter, we will discuss **constants** and **literals** in PL/SQL. A constant holds a value that once declared, does not change in the program. A constant declaration specifies its name, data type, and value, and allocates storage for it. The declaration can also impose the **NOT NULL constraint**.

Declaring a Constant

A constant is declared using the **CONSTANT** keyword. It requires an initial value and does not allow that value to be changed. For example −

```
PI CONSTANT NUMBER := 3.141592654;
DECLARE
   -- constant declaration
   pi constant number := 3.141592654;
   -- other declarations
   radius number(5,2);
   dia number(5,2);
   circumference number(7, 2);
   area number (10, 2);
```

```
BEGIN
   -- processing
   radius := 9.5;
   dia := radius * 2;
   circumference := 2.0 * pi * radius;
   area := pi * radius * radius;
   -- output
   dbms_output.put_line('Radius: ' || radius);
   dbms_output.put_line('Diameter: ' || dia);
   dbms_output.put_line('Circumference: ' || circumference);
   dbms_output.put_line('Area: ' || area);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
Radius: 9.5
Diameter: 19
Circumference: 59.69
Area: 283.53

Pl/SQL procedure successfully completed.
```

The PL/SQL Literals

A literal is an explicit numeric, character, string, or Boolean value not represented by an identifier. For example, TRUE, 786, NULL, 'tutorialspoint' are all literals of type Boolean, number, or string. PL/SQL, literals are case-sensitive. PL/SQL supports the following kinds of literals −

- Numeric Literals
- Character Literals
- String Literals
- BOOLEAN Literals
- Date and Time Literals

The following table provides examples from all these categories of literal values.

S.No	Literal Type & Example
1	**Numeric Literals** 050 78 -14 0 +32767 6.6667 0.0 -12.0 3.14159 +7800.00 6E5 1.0E-8 3.14159e0 -1E38 -9.5e-3

2	**Character Literals** 'A' '%' '9' ' ' 'z' '('
3	**String Literals** 'Hello, world!' 'Tutorials Point' '19-NOV-12'
4	**BOOLEAN Literals** TRUE, FALSE, and NULL.
5	**Date and Time Literals** DATE '1978-12-25'; TIMESTAMP '2012-10-29 12:01:01';

To embed single quotes within a string literal, place two single quotes next to each other as shown in the following program −

```
DECLARE
   message  varchar2(30):= 'That''s tutorialspoint.com!';
BEGIN
   dbms_output.put_line(message);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

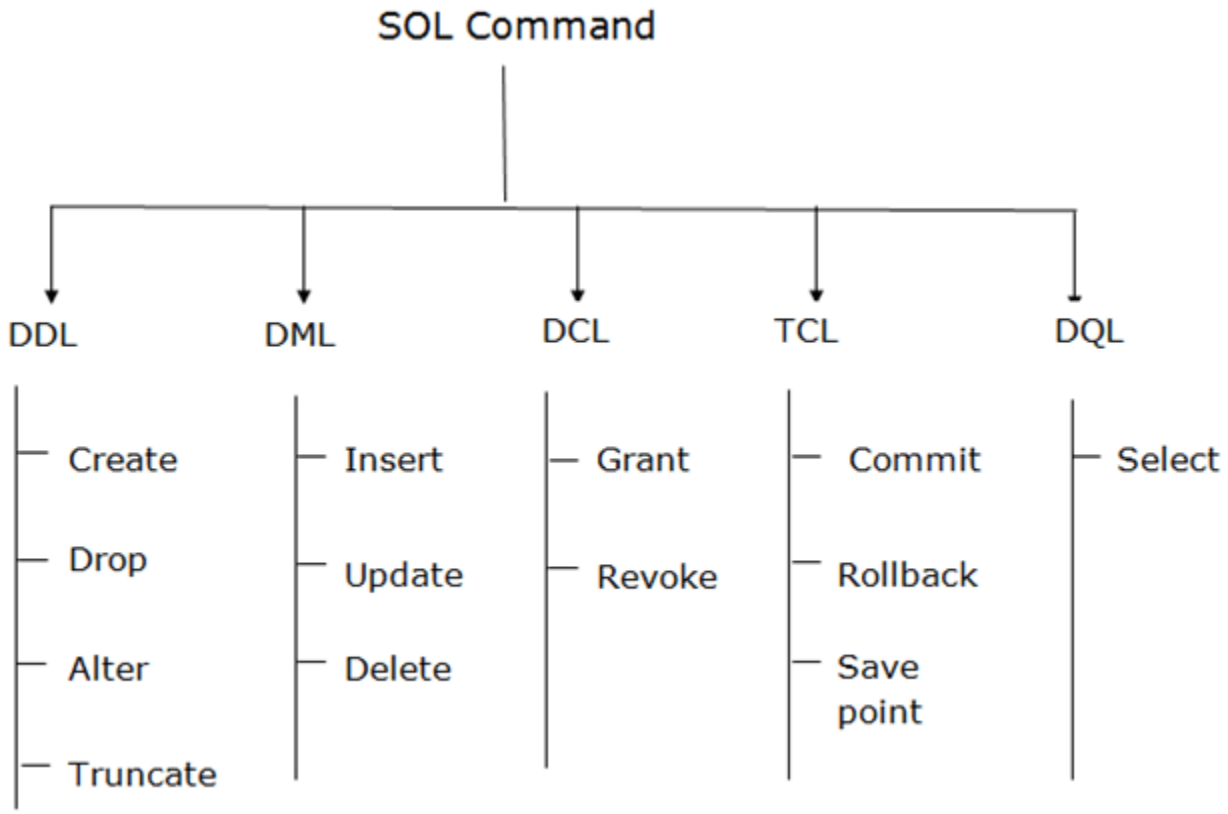
```
That's tutorialspoint.com!

PL/SQL procedure successfully completed.
```

SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, and set permission for users.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

a. CREATE It is used to create a new table in the database.

Syntax:

1. CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES [...]);

Example:

```
1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
```

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

```
1. DROP TABLE ;
```

Example

```
1. DROP TABLE EMPLOYEE;
```

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add a new column in the table

```
1. ALTER TABLE table_name ADD column_name COLUMN-definition;
```

To modify existing column in the table:

```
1. ALTER TABLE MODIFY(COLUMN DEFINITION....);
```

EXAMPLE

```
1. ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
2. ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));
```

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

```
1. TRUNCATE TABLE table_name;
```

Example:

```
1. TRUNCATE TABLE EMPLOYEE;
```

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

```
INSERT INTO TABLE_NAME
(col1, col2, col3,.... col N)
VALUES (value1, value2, value3, .... valueN);
```

Or

```
INSERT INTO TABLE_NAME   EMP_
VALUES (value1, value2, value3, .... valueN);
```

For example:

```
INSERT INTO javatpoint (Author, Subject) VALUES ("amit", "DBMS");
```

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

```
UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDIT
ION]
```

For example:

```
UPDATE students
SET User_Name = 'Sonoo'
WHERE Student_Id = '3'
```

c. DELETE: It is used to remove one or more row from a table.

Syntax:

```
DELETE FROM table_name [WHERE condition];
```

For example:

```
DELETE FROM javatpoint
WHERE Author="Sonoo";
```

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

```
GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
```

b. Revoke: It is used to take back permissions from the user.

Example

```
REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;
```

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

```
COMMIT;
```

Example:

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

```
ROLLBACK;
```

Example:

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
ROLLBACK;
```

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

```
SAVEPOINT SAVEPOINT_NAME;
```

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

```
1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;
```

For example:

```
1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;
```

SQL Operators | Arithmetic, Comparison & Logical Operators

As a SQL developer, often there are scenarios when you tend to manipulate and retrieve the data in a specific way. In other words, you would want to perform various operations. And knowing SQL operators is the key solution to all such requirements. So let us first understand

What is an Operator?

An operator in SQL is a reserved keyword or a symbol (special character) that operates up on the operands (or a set of operands) to perform various operations to return a result.

What is an Operand?

An operand can be defined as the data item or an argument that is used by an operator to perform different operations like arithmetic, logical, comparison, etc.

SQL Arithmetic Operators

Arithmetic operators operate on numeric operands. These are mainly used for mathematical operations such as addition, subtraction, etc.

There are two types of arithmetic operators as shown below:

1. **Unary Arithmetic Operators:**

These operators act upon only one operand. The format followed by the Unary operator is ‘OPERATOR OPERAND’.

2. **Binary Arithmetic Operators:**

These operators act upon two operands. The format followed by the Binary operator is 'OPERAND OPERATOR OPERAND'.

Unary Arithmetic Operators

	OPERATOR	OPERATION	EXPLANATION
1	+	UNARY POSITIVE	To make the operand or a value positive.
2	–	UNARY NEGATIVE	To negate the operand or make a value negative.

Binary Arithmetic Operators

	OPERATOR	OPERATION	EXPLANATION
3	+	ADDITION	Adds up the operand values specified on either side of the operator.
4	–	SUBTRACTION	Performs subtraction of right hand side value from the left hand side value.
5	*	MULTIPLICATION	Multiplies the operand values present on either side of operator.
6	/	DIVISION	Performs division of left hand side value by right hand side value and returns the quotient.
7	%	MODULUS	Performs division of left hand side value by right hand side value and returns the remainder.

For a better understanding of the usage of Arithmetic operators in the SQL queries, you can refer the examples provided below.

1.**Unary Positive (+)**

```
Select +10 as VALUE from dual;
```

Output:

VALUE
10

2. **Unary Negative (-)**

```
Select –20 as VALUE from dual;
```

Output:

VALUE
-20

3. **Addition (+)**

```
Select 10+20 as VALUE from dual;
```

Output:

VALUE
30

4. **Subtraction (-)**

Example 1:

```
Select 10-20 as VALUE from dual;
```

Output:

VALUE
-10

Example 2:

```
Select 20-10 as VALUE from dual;
```

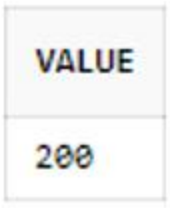
Output:

VALUE
10

5. **Multiplication (*)**

Select 10*****20 **as VALUE from** dual**;**

Output:



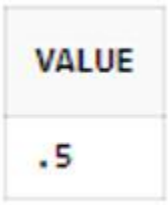
VALUE
200

6. **Division (/)**

Example 1:

Select 10/20 **as VALUE from** dual**;**

Output:

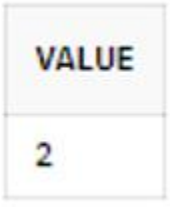


VALUE
.5

Example 2:

Select 20/10 **as VALUE from** dual**;**

Output:



VALUE
2

7. **Modulus (%)**

Example 1:

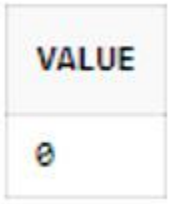
Select 10**%**20 **as VALUE from** dual**;**

Output:

Example 2:

Select 20**%**10 **as VALUE from** dual**;**

Output:



SQL Comparison Operators

Comparison operators compare two operand values or can also be used in conditions where one expression is compared to another that often returns a result (Result can be true or false).

	OPERATOR	OPERATION	EXPLANATION
1	=	Equality test	Performs the test on two operand values to check if they are equal. If they are equal, condition (in the where clause) becomes true and returns the corresponding rows.
2	<> or != or ^=	Inequality test	Performs the test on two operand values to check if they are not equal. If they are not equal, returns true else returns false.
3	>	Greater than test	Tests if the left hand side operand value is greater than that of operand value on the right hand side. If yes, returns true. Another case is if you use it in where clause of a select query, then it returns all those rows having column value greater than the specified value.
4	<	Less than test	Tests if the left hand side operand value is less than that of operand value on the right hand side. If yes, returns true. Another case where you can use it in where clause of a select query. In such a case, it returns all those rows having column values less than the specified value.
5	<=	Less than or equal to test	Evaluates if the left hand side operand value is less than or equal to the operand value on the right hand side. If yes, returns true.

6	>=	Greater than or equal to test	Evaluates if the left hand side operand value is greater than or equal to the operand value on the right hand side. If yes, returns true.

For a better understanding of the usage of comparison operators in the SQL queries, you can refer to the examples provided below.

Consider the table below which is being used as a reference for the next examples.

Sample Table – FACULTY_DETAILS

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
100	Diana	Lorentz	07-FEB-15	DLORENTZ@gmail.com	9834762482	JF	15000
101	James	Madison	23-MAR-17	JMadison@yahoo.com	8837482649	JF	16000
102	Ross	Lynch	27-OCT-13	RLYNC@gmail.com	9746827498	AF	18000
103	Nancy	Greenberg	13-DEC-12	NGREENB@gmail.com	7862438242	AF	20000
104	Lucy	Geller	05-SEP-09	LUC12@gmail.com	7826472642	SF	37000
105	Mark	Steven	01-JAN-10	MSTEVE@gmail.com	9637462874	SF	30000
106	Mike	Ferry	17-FEB-10	MFERRY@gmail.com	8386487264	SF	43000
107	Stefan	Jonaski	09-JAN-01	SJONASKk@gmail.com	9374824726	HOD	60000

1.**Equal to (=)**

Example 1: [Scenario of retrieving rows with the value having NUMBER datatype]

SELECT * FROM FACULTY_DETAILS **WHERE** FACULTY_ID = 103**;**

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
103	Nancy	Greenberg	13-DEC-12	NGREENB@gmail.com	7862438242	AF	20000

Example 2: [Scenario of retrieving rows with the value having VARCHAR2 datatype]

SELECT * FROM FACULTY_DETAILS **WHERE** FIRST_NAME = ‘MARK’**;**

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
105	Mark	Steven	01-JAN-10	MSTEVE@gmail.com	9637462874	SF	30000

2. **Not Equal to (<> or != or ^=)**

SELECT * FROM FACULTY_DETAILS **WHERE** SALARY **!=** 20000 OR

OR SELECT * FROM FACULTY_DETAILS **WHERE** SALARY <> 20000**;**

OR SELECT * FROM FACULTY_DETAILS **WHERE** SALARY ^= 20000**;**

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
102	Ross	Lynch	27-OCT-13	RLYNC@gmail.com	9746827498	AF	18000
104	Lucy	Geller	05-SEP-09	LUC12@gmail.com	7826472642	SF	37000
106	Mike	Ferry	17-FEB-10	MFERRY@gmail.com	8386487264	SF	43000
100	Diana	Lorentz	07-FEB-15	DLORENTZ@gmail.com	9834762482	JF	15000
101	James	Madison	23-MAR-17	JMadison@yahoo.com	8837482649	JF	16000
105	Mark	Steven	01-JAN-10	MSTEVE@gmail.com	9637462874	SF	30000
107	Stefan	Jonaski	09-JAN-01	SJONASKk@gmail.com	9374824726	HOD	60000

3. **Greater than (>)**

SELECT * FROM FACULTY_DETAILS **WHERE** SALARY > 20000**;**

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
104	Lucy	Geller	05-SEP-09	LUC12@gmail.com	7826472642	SF	37000
106	Mike	Ferry	17-FEB-10	MFERRY@gmail.com	8386487264	SF	43000
105	Mark	Steven	01-JAN-10	MSTEVE@gmail.com	9637462874	SF	30000
107	Stefan	Jonaski	09-JAN-01	SJONASKk@gmail.com	9374824726	HOD	60000

4. **Less than (<)**

SELECT * FROM FACULTY_DETAILS **WHERE** SALARY < 20000**;**

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
102	Ross	Lynch	27-OCT-13	RLYNC@gmail.com	9746827498	AF	18000
100	Diana	Lorentz	07-FEB-15	DLORENTZ@gmail.com	9834762482	JF	15000
101	James	Madison	23-MAR-17	JMadison@yahoo.com	8837482649	JF	16000

5. **Less than or equal to (<=)**

SELECT * FROM FACULTY_DETAILS **WHERE** SALARY <= 30000**;**

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
102	Ross	Lynch	27-OCT-13	RLYNC@gmail.com	9746827498	AF	18000
100	Diana	Lorentz	07-FEB-15	DLORENTZ@gmail.com	9834762482	JF	15000
101	James	Madison	23-MAR-17	JMadison@yahoo.com	8837482649	JF	16000
103	Nancy	Greenberg	13-DEC-12	NGREENB@gmail.com	7862438242	AF	20000
105	Mark	Steven	01-JAN-10	MSTEVE@gmail.com	9637462874	SF	30000

6. **Greater than or equal to (>=)**

SELECT * FROM FACULTY_DETAILS **WHERE** SALARY => 30000**;**

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
104	Lucy	Geller	05-SEP-09	LUC12@gmail.com	7826472642	SF	37000
106	Mike	Ferry	17-FEB-10	MFERRY@gmail.com	8386487264	SF	43000
105	Mark	Steven	01-JAN-10	MSTEVE@gmail.com	9637462874	SF	30000
107	Stefan	Jonaski	09-JAN-01	SJONASKk@gmail.com	9374824726	HOD	60000

SQL Logical Operators

Logical operators are the special type of operators that helps us to retrieve data from the database with even more specificity.

OPERATOR	EXPLANATION
ALL	ALL is used to compare a value with all other values in the set. This set of values can either be a list or the result of the subquery. ‘ALL’ has to be preceded with any of the comparison operators always. Suppose, a query returns no rows then it evaluates to be

	TRUE.
AND	AND permits the occurrence of multiple conditions in a SQL query. Only those rows are returned that satisfy all the conditions mentioned in a SQL statement.
ANY	ANY is used to compare a value with any of the values specified either in a list or in the result of the subquery. 'ANY' has to be preceded with any of the comparison operators always. Suppose, a query returns no rows then it evaluates to be FALSE.
BETWEEN 'x' and 'y'	It returns the rows which fall into a specified range where x is the minimum value and y is the maximum value (Both maximum and minimum values are included). It is equivalent to saying the range is greater than or equal to x and less than or equal to y.
NOT BETWEEN 'x' and 'y'	It returns the rows which do not fall into a specified range where x is the minimum value and y is the maximum value. It is equivalent to saying the range is not greater than or equal to x and not less than or equal to y.
IN	IN is equivalent to '=ANY'. It compares a particular value to another set of values mentioned in list separated by commas or a subquery that returns the set of values. If matches are found, then it returns all such rows.
NOT IN	IN is equivalent to '!=ANY'. It compares a particular value to another set of values mentioned in a list separated by commas or a subquery that returns the set of values. If matches are found, then it returns all other rows except the ones for which match has been found.
LIKE	LIKE operator uses the wildcard notations '%' and '_' to search for specific values in the column. '%' represents zero, one or multiple characters and '_' represents a single character. Note that '*' represents zero, one, or multiple characters where as '?' represents a single character in MS Access.
OR	OR permits the occurrence of multiple conditions in a SQL query. All those rows are returned which satisfy any of the conditions mentioned in a SQL statement.
SOME	SOME is used to compare a value with any of the values specified either in a list or in the result of the subquery. 'SOME' has to be preceded with any of the comparison operators always. This operator functions similar to ANY.
IS NULL	This operator is used when there is a need to deal with NULL values. We can specify this as a condition in where clause of the SQL query which returns the rows (if present) having column value equal to NULL.
IS NOT NULL	It can be specified in a where clause of a SQL query which then returns the rows that do not have nulls.
EXISTS	It compares a specified column value with that of column values present in another table or a subquery result along with the other mentioned criteria. If matches are found, it returns all such rows.

NOT EXISTS	It compares a specified column value with that of column values present in another table or a subquery result along with the other mentioned criteria. It returns all other rows except the ones that do exist as a result of the subquery.

For a better understanding of the usage of Logical operators in SQL queries, you can check out the examples provided below.

Consider the tables “FACULTY_DETAILS” and “JOBS” below that is being used as a reference for the next examples.

Sample Table 1 – FACULTY_DETAILS

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
100	Diana	Lorentz	07-FEB-15	DLORENTZ@gmail.com	9834762482	JF	15000
101	James	Madison	23-MAR-17	JMadison@yahoo.com	8837482649	JF	16000
102	Ross	Lynch	27-OCT-13	RLYNC@gmail.com	9746827498	AF	18000
103	Nancy	Greenberg	13-DEC-12	NGREENB@gmail.com	7862438242	AF	20000
104	Lucy	Geller	05-SEP-09	LUC12@gmail.com	7826472642	SF	37000
105	Mark	Steven	01-JAN-10	MSTEVE@gmail.com	9637462874	SF	30000
106	Mike	Ferry	17-FEB-10	MFERRY@gmail.com	8386487264	SF	43000
107	Stefan	Jonaski	09-JAN-01	SJONASKk@gmail.com	9374824726	HOD	60000

Sample Table 2 – JOBS

JOB_ID	DESIGNATION	TYPE_OF_JOB	WORKING_HOURS
JF	Junior faculty	PART TIME	4
SF	Senior faculty	FULL TIME	7
HOD	Head of Department	FULL TIME	8
AF	Assitant Professor	FULL TIME	7

1. **ALL**

Example 1:

SELECT * FROM FACULTY_DETAILS **WHERE** SALARY > **ALL (SELECT** SALARY **FROM** FACU LTY_DETAILS **WHERE** FACULTY_ID = 104**);**

Referring to the above statement, we know that the subquery returns one record with SALARY = 37000. So, the main query returns all such records whose SALARY is greater than 37000.

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
106	Mike	Ferry	17-FEB-10	MFERRY@gmail.com	8386487264	SF	43000
107	Stefan	Jonaski	09-JAN-01	SJONASKk@gmail.com	9374824726	HOD	60000

Example 2:

SELECT * FROM FACULTY_DETAILS **WHERE** SALARY > **ALL (SELECT** SALARY **FROM** FACU LTY_DETAILS **WHERE** FACULTY_ID < 104**);**

This time the subquery returns four records (with SALARY = 15000, 16000, 18000, 20000). So, the main query returns all such records whose SALARY is greater than all of those returned from the subquery.

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
105	Mark	Steven	01-JAN-10	MSTEVE@gmail.com	9637462874	SF	30000
104	Lucy	Geller	05-SEP-09	LUC12@gmail.com	7826472642	SF	37000
106	Mike	Ferry	17-FEB-10	MFERRY@gmail.com	8386487264	SF	43000
107	Stefan	Jonaski	09-JAN-01	SJONASKk@gmail.com	9374824726	HOD	60000

Example 3:

SELECT * FROM FACULTY_DETAILS **WHERE** SALARY < **ALL (SELECT** SALARY **FROM** FACU LTY_DETAILS **WHERE** FACULTY_ID > 104**);**

In this case, the subquery returns three records (with SALARY = 30000, 43000, 60000). So, the main query returns all such records whose SALARY is less than all of those returned from the subquery.

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
103	Nancy	Greenberg	13-DEC-12	NGREENB@gmail.com	7862438242	AF	20000
102	Ross	Lynch	27-OCT-13	RLYNC@gmail.com	9746827498	AF	18000
101	James	Madison	23-MAR-17	JMadison@yahoo.com	8837482649	JF	16000
100	Diana	Lorentz	07-FEB-15	DLORENTZ@gmail.com	9834762482	JF	15000

2. **AND**

SELECT * FROM FACULTY_DETAILS **WHERE** FACULTY_ID = 104 **AND** JOB_ID = 'SF'**;**

In the above statement, there are two conditions in the where clause. AND Operator makes sure that only those rows are returned that satisfy both the conditions. Likewise, you can specify as many conditions as you want.

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
104	Lucy	Geller	05-SEP-09	LUC12@gmail.com	7826472642	SF	37000

3. **ANY**

Example 1:

Here, the subquery returns three records (with SALARY = 30000, 43000, 60000). So, the main query returns all such records whose SALARY is greater than any one value of those returned from the subquery.

SELECT * FROM FACULTY_DETAILS **WHERE** SALARY > **ANY (SELECT** SALARY **FROM** FACU LTY_DETAILS **WHERE** FACULTY_ID > 104**);**

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
107	Stefan	Jonaski	09-JAN-01	SJONASKk@gmail.com	9374824726	HOD	60000
106	Mike	Ferry	17-FEB-10	MFERRY@gmail.com	8386487264	SF	43000
104	Lucy	Geller	05-SEP-09	LUC12@gmail.com	7826472642	SF	37000

Example 2:

SELECT * FROM FACULTY_DETAILS **WHERE** SALARY = **ANY (SELECT** SALARY **FROM** FACU LTY_DETAILS **WHERE** FACULTY_ID > 104**);**

In this scenario, the subquery returns three records (with SALARY = 30000, 43000, 60000). So, the main query returns all such records which are returned as a result of the subquery.

Output

FACULTY_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	EMAIL	PHONE_NUMBER	JOB_ID	SALARY
105	Mark	Steven	01-JAN-10	MSTEVE@gmail.com	9637462874	SF	30000
106	Mike	Ferry	17-FEB-10	MFERRY@gmail.com	8386487264	SF	43000
107	Stefan	Jonaski	09-JAN-01	SJONASKk@gmail.com	9374824726	HOD	60000

Example 3:

SELECT * FROM FACULTY_DETAILS **WHERE** SALARY < **ANY (SELECT** SALARY **FROM** FACU LTY_DETAILS **WHERE** FACULTY_ID > 105**);**

Output

UNIT-4

Inclusion Dependency

- Multivalve dependency and join dependency can be used to guide database design although they both are less common than functional dependencies.
- Inclusion dependencies are quite common. They typically show little influence on designing of the database.
- The inclusion dependency is a statement in which some columns of a relation are contained in other columns.
- The example of inclusion dependency is a foreign key. In one relation, the referring relation is contained in the primary key column(s) of the referenced relation.
- Suppose we have two relations R and S which was obtained by translating two entity sets such that every R entity is also an S entity.

- Inclusion dependency would be happen if projecting R on its key attributes yields a relation that is contained in the relation obtained by projecting S on its key attributes.
- In inclusion dependency, we should not split groups of attributes that participate in an inclusion dependency.
- In practice, most inclusion dependencies are key-based that is involved only keys.

Decomposition

Decomposition of a relation is done when a relation in relational model is not in appropriate normal form. Relation R is decomposed into two or more relations if decomposition is lossless join as well as dependency preserving.

Decomposition is of two types:

1. Lossless join Decomposition
2. Loss Decomposition

Lossless Join Decomposition

- Consider there is a relation R which is decomposed into sub relations R1, R2, , Rn.
- This decomposition is called lossless join decomposition when the join of the sub relations results in the same relation R that was decomposed.
- For lossless join decomposition, we always have- R1 ⋈ R2 ⋈ R3 ……. ⋈ Rn = R , where ⋈ is a natural join operator

Example: Consider the following relation R(A , B , C)-

A	B	C
1	2	1
2	5	3
3	3	3

R(A , B , C)

Consider this relation is decomposed into two sub relations R1 (A , B) and R2(B , C)-

- R(A,B,C)

1. R1(A,B)
2. R2(B,C)

The two sub relations are-

A	B
1	2
2	5
3	3

R1 (A, B)

B	C
2	1
5	3

B	C
3	3

R2(B , C)

Now, let us check whether this decomposition is lossless or not.

For lossless decomposition, we must have-

R1 ⋈ R2 = R

Now, if we perform the natural join (⋈) of the sub relations R1 and R2 , we get-

A	B	C
1	2	1
2	5	3
3	3	3

This relation is same as the original relation R.

Thus, we conclude that the above decomposition is lossless join decomposition.

NOTE

- Lossless join decomposition is also known as non-additive join decomposition.
- This is because the resultant relation after joining the sub relations is same as the decomposed relation.
- No extraneous tuples appear after joining of the sub-relations.

Loss Join Decomposition:

- Consider there is a relation R which is decomposed into sub relations R1 , R2 , …. , Rn.
- This decomposition is called lossy join decomposition when the join of the sub relations does not result in the same relation R that was decomposed.
- The natural join of the sub relations is always found to have some extraneous tuples.
- For lossy join decomposition, we always have- R1 ⋈ R2 ⋈ R3 ……. ⋈ Rn ⊃ R where ⋈ is a natural join operator

Example : Consider that we have table STUDENT with three attribute roll_no , sname and department.

Student		
Roll_no	**Sname**	**Dept**
111	parimal	COMPUTER
222	parimal	ELECTRICAL

This relation is decomposed into two relation no_name and name_dept :

No_name	
Roll_no	**Sname**
111	parimal

No_name	
Roll_no	**Sname**
222	parimal

name_dept	
Sname	**Dept**
parimal	COMPUTER
parimal	ELECTRICAL

In loss decomposition, spurious tuples are generated when a natural join is applied to the relations in the decomposition.

stu_joined		
Roll_no	**Sname**	**Dept**
111	parimal	COMPUTER
111	parimal	ELECTRICAL
222	parimal	COMPUTER
222	parimal	ELECTRICAL

The above decomposition is a bad decomposition or Lossy decomposition.

UNIT-5

TRANSACTION PROCESSING CONCEPT

Transaction

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

1. Open_Account(X)

2. Old_Balance = X.balance
3. New_Balance = Old_Balance - 800
4. X.balance = New_Balance
5. Close_Account(X)

Y's Account

1. Open_Account(Y)
2. Old_Balance = Y.balance
3. New_Balance = Old_Balance + 800
4. Y.balance = New_Balance
5. Close_Account(Y)

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. 1. R(X);
2. 2. X = X - 500;
3. 3. W(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

Transaction property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durability

Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2
Read(A) A:= A-100 Write(A)	Read(B) Y:= Y+100 Write(B)

After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

For example: The total amount must be maintained before or after the transaction.

1. Total before T occurs = 600+300=900
2. Total after T occurs= 500+400=900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.

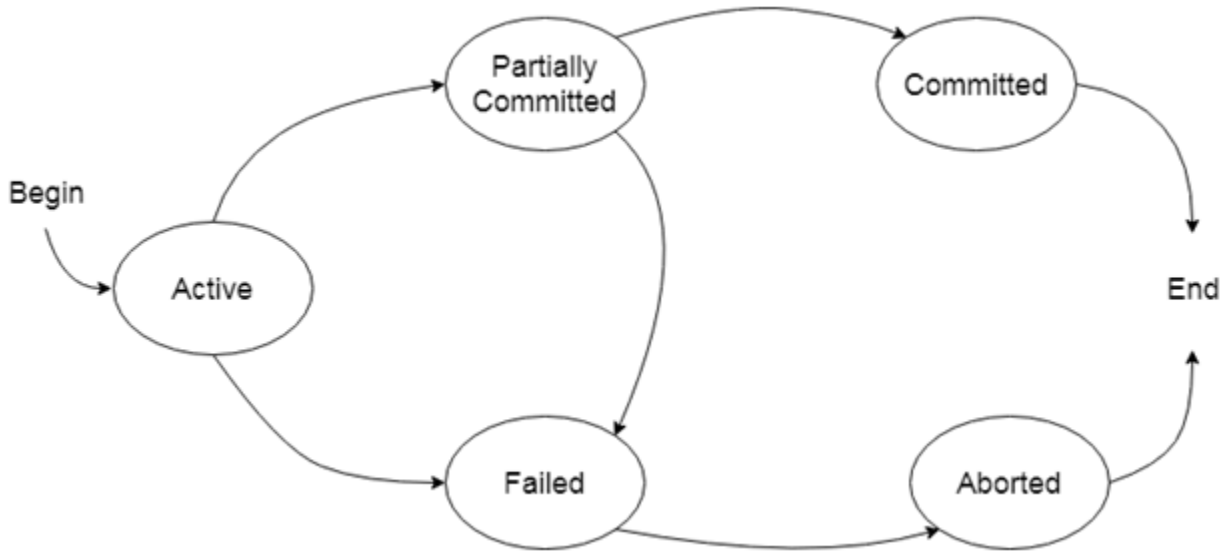
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transaction

In a database, the transaction can be in one of the following states -



Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.

- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

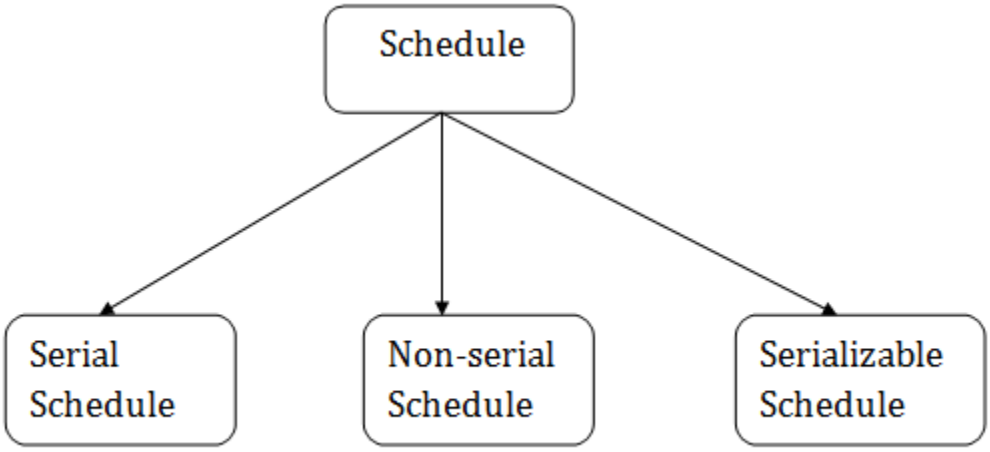
- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    1. Re-start the transaction
    2. Kill the transaction

Schedule

A series of operation from one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transaction.



1. Serial Schedule

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

1. Execute all the operations of T1 which was followed by all the operations of T2.
2. Execute all the operations of T1 which was followed by all the operations of T2.

- In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.
- In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

2. Non-serial Schedule

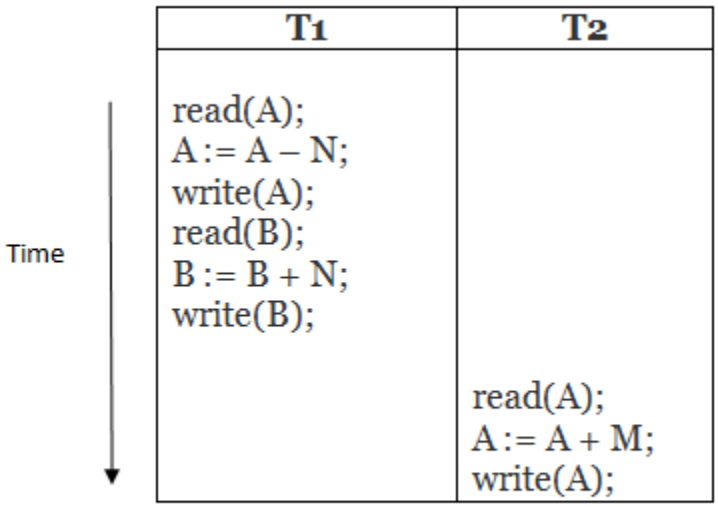
- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

3. Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.

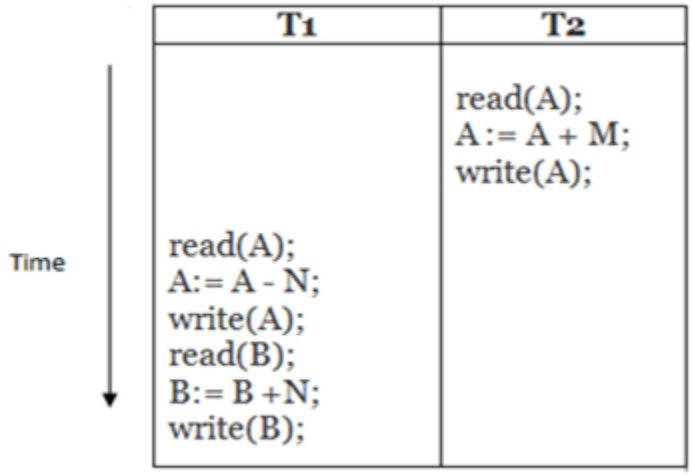
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

(a)



Schedule A

(b)



Schedule B

(c)

Time

T1	T2
read(A); A := A – N;	
	read(A); A := A + M;
write(A); read(B);	
	write(A);
B := B + N; write(B);	

Schedule C

(d)

Time

T1	T2

read(A);
A := A – N;
write(A);	

	read(A);
	A := A + M;
	write(A);

read(B);
B := B + N;
write(B);	

Schedule D

Here,

Schedule A and Schedule B are serial schedule.

Schedule C and Schedule D are Non-serial schedule.

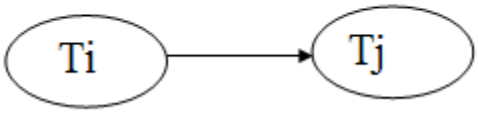
Testing of Serializability

Serialization Graph is used to test the Serializability of a schedule.

Assume a schedule S. For S, we construct a graph known as precedence graph. This graph has a pair G = (V, E), where V consists a set of vertices, and E consists a set of edges. The set of vertices is used to contain all the transactions participating in the schedule. The set of edges is used to contain all edges Ti ->Tj for which one of the three conditions holds:

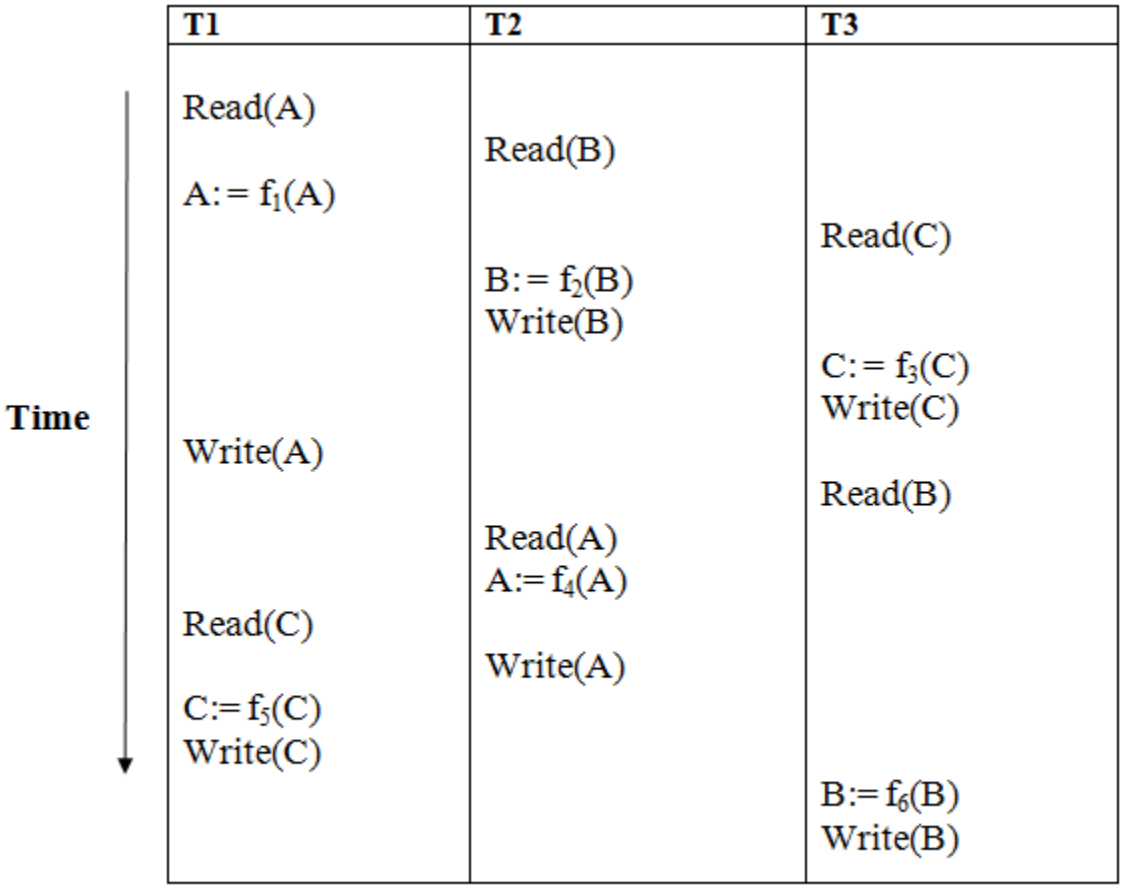
1. Create a node Ti → Tj if Ti executes write (Q) before Tj executes read (Q).
2. Create a node Ti → Tj if Ti executes read (Q) before Tj executes write (Q).
3. Create a node Ti → Tj if Ti executes write (Q) before Tj executes write (Q).

Precedence graph for Schedule S



- If a precedence graph contains a single edge Ti → Tj, then all the instructions of Ti are executed before the first instruction of Tj is executed.
- If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

For example:

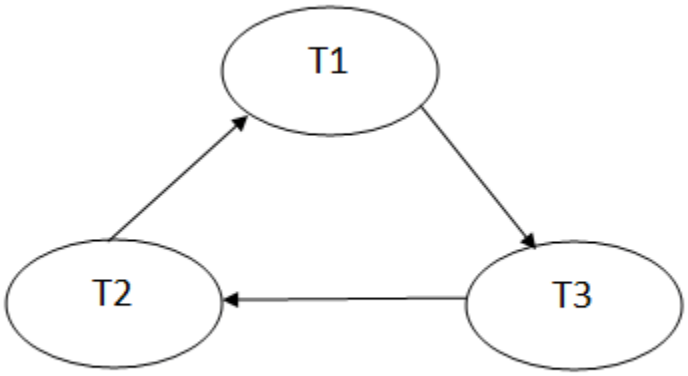


Schedule S1

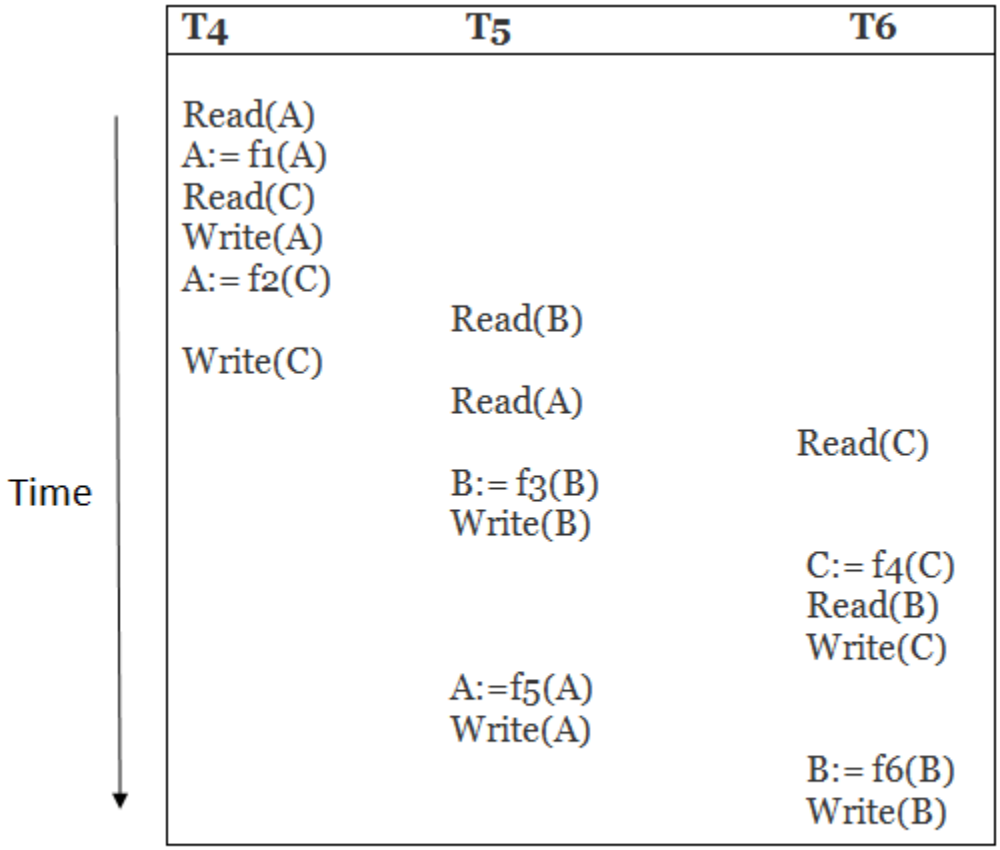
Explanation:

Read(A): In T1, no subsequent writes to A, so no new edges
Read(B): In T2, no subsequent writes to B, so no new edges
Read(C): In T3, no subsequent writes to C, so no new edges
Write(B): B is subsequently read by T3, so add edge T2 → T3
Write(C): C is subsequently read by T1, so add edge T3 → T1
Write(A): A is subsequently read by T2, so add edge T1 → T2
Write(A): In T2, no subsequent reads to A, so no new edges
Write(C): In T1, no subsequent reads to C, so no new edges
Write(B): In T3, no subsequent reads to B, so no new edges

Precedence graph for schedule S1:



The precedence graph for schedule S1 contains a cycle that's why Schedule S1 is non-serializable.

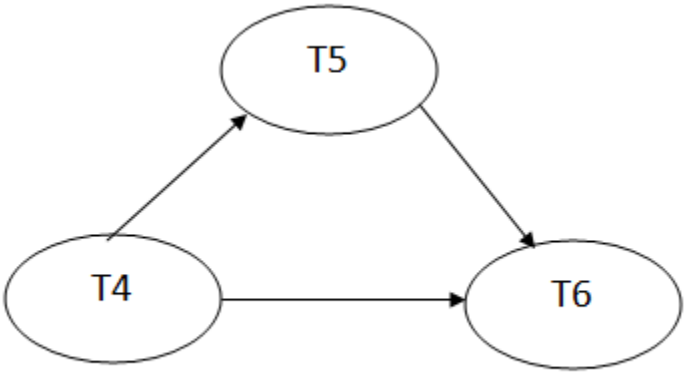


Schedule S2

Explanation:

Read(A): In T4,no subsequent writes to A, so no new edges
Read(C): In T4, no subsequent writes to C, so no new edges
Write(A): A is subsequently read by T5, so add edge T4 → T5
Read(B): In T5,no subsequent writes to B, so no new edges
Write(C): C is subsequently read by T6, so add edge T4 → T6
Write(B): A is subsequently read by T6, so add edge T5 → T6
Write(C): In T6, no subsequent reads to C, so no new edges
Write(A): In T5, no subsequent reads to A, so no new edges
Write(B): In T6, no subsequent reads to B, so no new edges

Precedence graph for schedule S2:



The precedence graph for schedule S2 contains no cycle that's why ScheduleS2 is serializable.

Conflict Serializable Schedule

- A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
- The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

Conflicting Operations

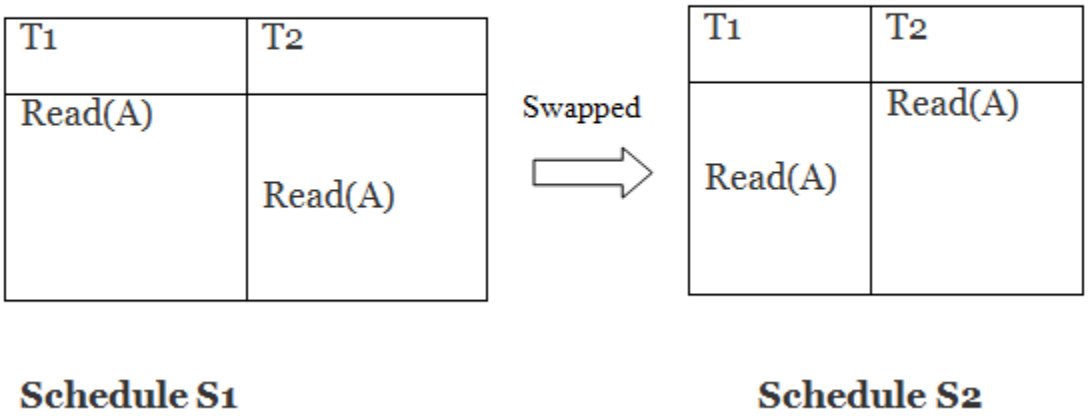
The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Example:

Swapping is possible only if S1 and S2 are logically equal.

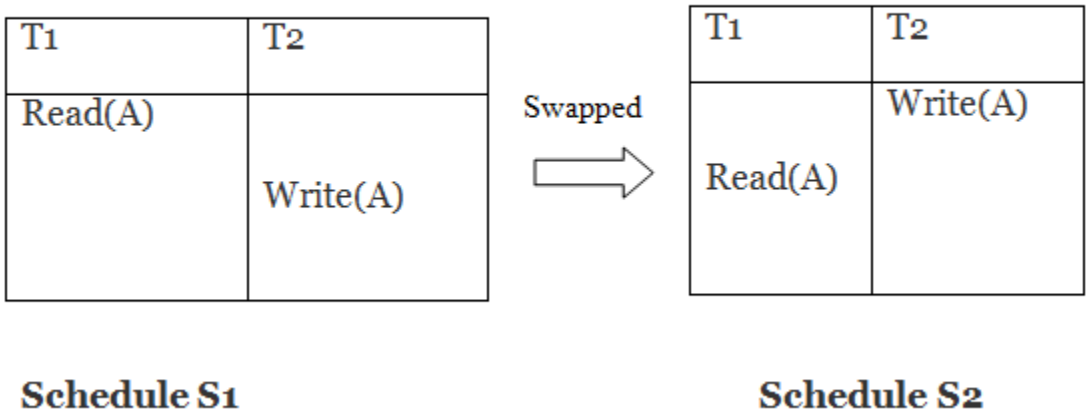
1. T1: Read(A) T2: Read(A)



Schedule S1 **Schedule S2**

Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)



Schedule S1 **Schedule S2**

Here, S1 ≠ S2. That means it is conflict.

Conflict Equivalent

In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations. In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations).

Two schedules are said to be conflict equivalent if and only if:

1. They contain the same set of the transaction.

2. If each pair of conflict operations are ordered in the same way.

Example:

Non-serial schedule

T1	T2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Schedule S1

Serial Schedule

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S2

Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2. Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.

After swapping of non-conflict operations, the schedule S1 becomes:

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Since, S1 is conflict serializable.

View Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which d.oes not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	
	Write(A)

Schedule S1

T1	T2
	Write(A)
Read(A)	

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)

Schedule S1

T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

Schedule S2

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)		
	Read(A)	
		Write(A)

Schedule S1

T1	T2	T3
	Read(A)	
Write(A)		
		Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Example:

T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)

Schedule S

With 3 transactions, the total number of possible schedule

1. = 3! = 6
2. S1 = <T1 T2 T3>
3. S2 = <T1 T3 T2>
4. S3 = <T2 T3 T1>
5. S4 = <T2 T1 T3>
6. S5 = <T3 T1 T2>
7. S6 = <T3 T2 T1>

Taking first schedule S1:

T1	T2	T3
Read(A) Write(A)	Write(A)	Write(A)

Schedule S1

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is:

1. T1 → T2 → T3

Recoverability of Schedule

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

The above table 1 shows a schedule which has two transactions. T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1. T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule.

Irrecoverable schedule: The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

The above table 2 shows a schedule with two transactions. Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1. T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is recoverable with cascade rollback.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

The above Table 3 shows a schedule with two transactions. Transaction T1 reads and write A and commits, and that value is read and written by T2. So this is a cascade less recoverable schedule.

Failure Classification

To find that where the problem has occurred, we generalize a failure into the following categories:

1. Transaction failure

2. System crash
3. Disk failure

1. Transaction failure

The transaction failure occurs when it fails to execute or when it reaches a point from where it can't go any further. If a few transaction or process is hurt, then this is called as transaction failure.

Reasons for a transaction failure could be -

1. **Logical errors:** If a transaction cannot complete due to some code error or an internal error condition, then the logical error occurs.
2. **Syntax error:** It occurs where the DBMS itself terminates an active transaction because the database system is not able to execute it. **For example,** The system aborts an active transaction, in case of deadlock or resource unavailability.

2. System Crash

- System failure can occur due to power failure or other hardware or software failure. **Example:** Operating system error.

  Fail-stop assumption: In the system crash, non-volatile storage is assumed not to be corrupted.

3. Disk Failure

- It occurs where hard-disk drives or storage drives used to fail frequently. It was a common problem in the early days of technology evolution.
- Disk failure occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all or part of disk storage.

Log-Based Recovery

- The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
- If any operation is performed on the database, then it will be recorded in the log.
- But the process of storing the logs should be done before the actual transaction is applied in the database.

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.

1. <Tn, Start>

- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

1. <Tn, City, 'Noida', 'Bangalore' >

- When the transaction is finished, then it writes another log to indicate the end of the transaction.

1. <Tn, Commit>

There are two approaches to modify the database:

1. Deferred database modification:

- The deferred modification technique occurs if the transaction does not modify the database until it has committed.
- In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

2. Immediate database modification:

- The Immediate modification technique occurs if database modification occurs while the transaction is still active.
- In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Recovery using Log records

When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

1. If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then the Transaction Ti needs to be redone.
2. If log contains record<T_n, Start> but does not contain the record either <Ti, commit> or <Ti, abort>, then the Transaction Ti needs to be undone.

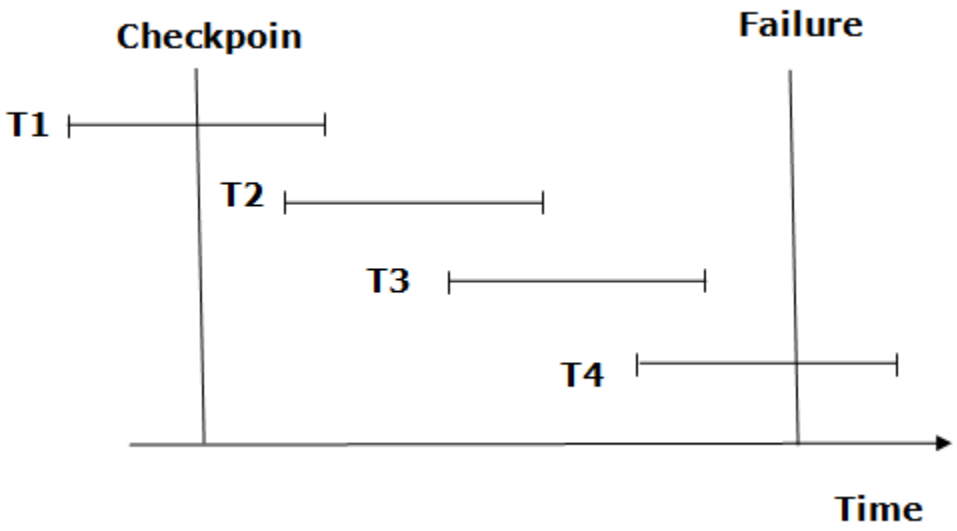
Checkpoint

- The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk.
- The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log files will be created.

- When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on.
- The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed.

Recovery using Checkpoint

In the following manner, a recovery system recovers the database from this failure:



- The recovery system reads log files from the end to start. It reads log files from T4 to T1.
- Recovery system maintains two lists, a redo-list, and an undo-list.
- The transaction is put into redo state if the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.
- **For example:** In the log file, transaction T2 and T3 will have <Tn, Start> and <Tn, Commit>. The T1 transaction will have only <Tn, commit> in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.
- The transaction is put into undo state if the recovery system sees a log with <Tn, Start> but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.
- **For example:** Transaction T4 will have <Tn, Start>. So T4 will be put into undo list since this transaction is not yet complete and failed amid.

Deadlock in DBMS

A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.

For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

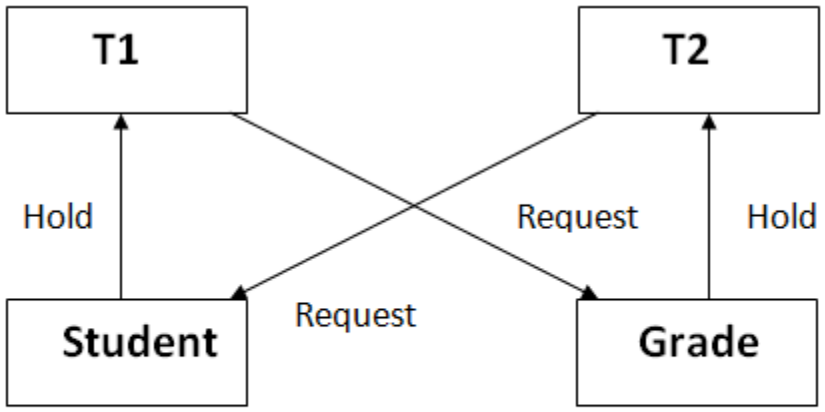


Figure: Deadlock in DBMS

Deadlock Avoidance

- When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.
- Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database. For the larger database, deadlock prevention method can be used.

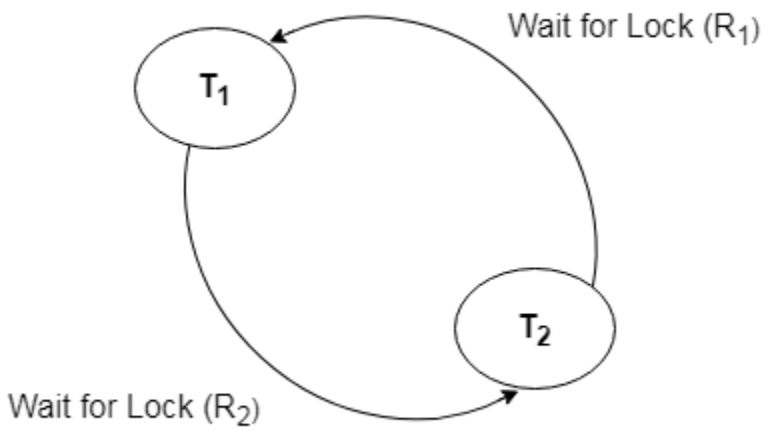
Deadlock Detection

In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

Wait for Graph

- This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.
- The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.

The wait for a graph for the above scenario is shown below:



Deadlock Prevention

- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
- The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.

Wait-Die scheme

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

Let's assume there are two transactions Ti and Tj and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:

1. Check if TS(Ti) < TS(Tj) - If Ti is the older transaction and Tj has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.
2. Check if TS(T_i) < TS(Tj) - If Ti is older transaction and has held some resource and if Tj is waiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.

Wound wait scheme

- In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted but with the same timestamp.
- If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.