(22CA102002) DATABASE MANAGEMENT SYSTEMs

Module 1: INTRODUCTION TO DATABASE SYSTEMS AND DATABASEDESIGN (08 Periods)

Introduction to Database Systems: Database system applications, Purpose of database systems,View of data - Data abstraction, Instances and schemas, Data models; Database languages - Data Definition Language, Data Manipulation Language; Database architecture, Database users and administrators.

Introduction to Database Design: Database design and ER diagrams, Entities, attributes and entity sets, Relationships and relationship sets, Additional features of ER model, Conceptual Design with ER model.

Module 2: RELATIONAL MODEL AND RELATIONAL ALGEBRA (08 Periods)

Relational Model: Creating and modifying relations, Integrity constraints over relations, Enforcing integrity constraints, Querying relational data, Logical database design, Introduction to views, Destroying/altering tables and views.

Relational Algebra: Preliminaries, Relational Algebra operators.

Module 3: SQL AND PL/SQL (10 Periods)

SQL: Form of basic SQL query, Nested queries, Aggregate operators, Null values, Complex integrityconstrains in SQL, Triggers and active databases.

PL/SQL: Generic PL/SQL block, PL/SQL data types, Control structure, Procedures and functions, Cursors, Database triggers.

Module 4: SCHEMA REFINEMENT AND TRANSACTIONS (10 Periods)

Schema Refinement: Problems caused by redundancy, Decompositions, Problems related to decomposition, Functional dependencies, Reasoning about FDs, First normal form, Second normal form, Third normal form, Boyce-Codd normal form, Multivalued dependencies, Fourth normal form, Join dependencies, Fifth normal form.

Transactions: Transaction concept, Transaction atomicity and durability, Concurrent Executions – Serializability, Recoverability, Implementation of isolation, Testing for serializability.

Module 5: CONCURRENCY CONTROL, STORAGE AND INDEXING (09 Periods)

Concurrency Control: Lock Based Protocols, Timestamp Based Protocols, Validation Based Protocols, Multiple Granularity, Deadlock Handling.

Storage and Indexing: Data on external storage, File organizations and indexing – Clustered indexes, Primary and secondary indexes; Index data structures – Hash based indexing, Tree based indexing; Comparison of file organizations.

Module 1

INTRODUCTION TO DATABASE SYSTEMS AND DATABASEDESIGN (08 Periods)

Introduction to Database Systems: Database system applications, Purpose of database systems,View of data - Data abstraction, Instances and schemas, Data models; Database languages - Data Definition Language, Data Manipulation Language; Database architecture, Database users and administrators.

Introduction:

Data: Data are raw facts. The word raw indicates that the facts have not yet been processed to reveal their meaning.

Or

The term data referred to known raw facts. Text, graphics, images and videosegments that have meaning in the user's environment

Information: Information is the result of processing raw data to reveal ors meaning. Data processing can be as simple as organizing data to reveal patterns or as complex as making forecasts or drawing inferences using statistical modeling.

Or

Data that have been processed in such a way as to increase the knowledge of theperson who use the data. Information is produced by processing data. Information is used to reveal the meaning of data. Accurate, relevant, and timely information os the key

Database: The database consists of logically related data stored in a single datarepository.

Metadata: The metadata provide a description of data characteristics and the set ofrelationships that link the data found with in the database.

Ex:

Name	Type	Length	Min	Max	Description
Course	Alphanumeric	30characters	-	-	Course ID and name

Database management system (DBMS):DBMS is collection of programs thatmanages the database structure and control access to the data stored in the database.

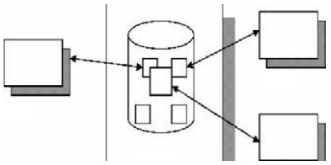
Field: A character or group of characters that has a specific meaning. A field is used todefine and store data.

Record: A logically connected a set of one or more fields that describes a person, placeor thing.

File: A collection of related records.

1 What is a Database Management System

1. A DBMS is a collection programs
2. That manages database structure
3. And control access to the data stored in the data base.



Or

1. A DBMS is a collection of programs
2. That enables to user to create, maintain the database.

A DBMS consists of:

1. A collection of interrelated and persistent data. This part of DBMS is referred to as database (DB).
2. A set of application programs used to access, update, and manage data. It iscalled data management system (DMS).
3. A DBMS is general-purpose software i.e., not application specific. The same DBMS(e.g., Oracle, Sybase, MS Acess, INFORMATICA ,MYSQL etc.)
4. It can be used in railway reservation system, library management, university, etc.
5. A DBMS takes care of storing and accessing data, leaving only application specifictasks to application programs.

2 Explain Structure of DBMS

1. The DBMS uses an application specific database description to define this translation.
2. The database description is generated by a database designer from his or her conceptual view of the database, which is called the Conceptual Schema.
3. The translation from the conceptual schema to the database description is performed using a data definition language (DDL) or a graphical or textual design interface.

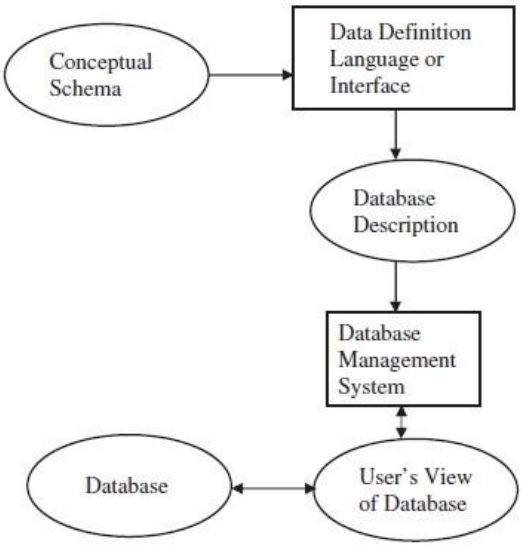


Fig. 1.3. Structure of database management system

➪ 3 What are the Objectives of DBMS

The main objectives of database management system are

1. Data availability,
2. Data integrity,
3. Data security, and
4. Data independence

1. Data Availability : Data availability refers to the fact That the data are creates available to thewide variety of users can access the date easily in a meaningful format

2. Data Integrity : Data integrity refers to the correctness of the data in the database.In other words, the data available in the database is a reliable data

3. Data Security : Data security refers to the fact that only authorized users can access the data.Data security can be enforced by passwords.

If two separate users are accessing a particular data at the same time, The DBMS must not allow them to make conflicting changes.

4. Data Independence : DBMS allows the user to store, update, and retrieve data in efficient manner.DBMS provides an "abstract view" of how the data is stored in the database.

In order to store the information efficiently, complex data structures are usedto represent the data.
The system hides certain details of how the data are stored and maintained.

⇨ **4 Explain Evolution of Database Management Systems**

In recent years, two approaches to DBMS are more popular, which are

- Object-Oriented DBMS (OODBMS) and
- Object Relational DBMS (ORDBMS).

The sequential order of the development of DBMS is as follows:

1. Flat files – 1960s–1980s
2. Hierarchical – 1970s–1990s
3. Network – 1970s–1990s
4. Relational – 1980s–present
5. Object-oriented – 1990s–present
6. Object-relational – 1990s–present
7. Data warehousing – 1980s–present
8. Web-enabled – 1990s–present

Early 1960s.

1. Charles Bachman at GE created the first general purpose DBMS IntegratedData Store.
2. It created the basis for the network model which was standardized byCODASYL (Conference on Data System Language).

Late 1960s:

1. IBM developed the Information Management System (IMS).
2. IMS used an alternate model, called the Hierarchical Data Model.

In 1970:

1. Edgar Codd, from IBM created the Relational Data Model.

In 1981:

1. Codd received the Turing Award for his contributions to database theory.
2. Codd Passed away in April 2003.

In 1976:
Peter Chen presented Entity-Relationship model, which is widely used in databasedesign.

In 1980:

1. SQL developed by IBM, became the standard query language for databases. SQLwas standardized by ISO.

In 1980s and 1990s:

1. IBM, Oracle, Informix and others developed powerful DBMS.

DATABASE SYSTEM APPLICATIONS

The Database Management System (DBMS) is defined as a software system that allows the user to define, create and maintain the database and provide control access to the data.

It is a collection of programs used for managing data and simultaneously it supports different types of users to create, manage, retrieve, update and store information.

In so many fields, we will use a database management system.

some of the applications where database management system uses −

Applications where we use Database Management Systems are:

- **Telecom**: There is a database to keeps track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.
- **Industry**: Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs. For example distribution centre should keep a track of the product units that supplied into the centre as well as the products that got delivered out from the distribution centre on each day; this is where DBMS comes into picture.
- **Banking System**: For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems. Also, banking system needs security of data as the data is sensitive, this is efficiently taken care by the DBMS systems.
- **Sales**: To store customer information, production information and invoice details. Using DBMS, you can track, manage and generate historical data to analyse the sales data.
- **Airlines**: To travel though airlines, we make early reservations, this reservation information along with flight schedule is stored in database. This is where the real-time update of data is necessary **as a flight seat reserved for one passenger should not be allocated to another passenger**, this is easily handled by the DBMS systems as the data updates are in real time and fast.
- **Education sector**: Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is a large amount of inter-related data that needs to be stored and retrieved in an efficient manner.
- **Online shopping**: You must be aware of the online shopping websites such as Amazon, Flipkart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system. **Along with managing the vast catalogue of items, there is a need to secure the user private information such as bank & card details**. All this is taken care of by database management systems.

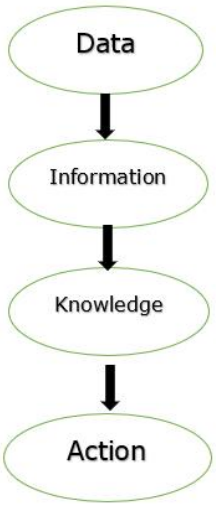
PURPOSE OF DATABASE SYSTEMS

The purpose of DBMS is to transform the following −

- Data into information.
- Information into knowledge.
- Knowledge to the action.

The diagram given below explains the process as to how the transformation of data to information to knowledge to action happens respectively in the DBMS −

Previously, the database applications were built directly on top of the file system



DBMS provides many advantages such as:

- Improved data sharing
- Improved data security
- Better data integration
- Minimized data inconsistency
- Improved data access
- Improved decision making
- Increased end-users productivity
- Mutli user data access
- Backup and recovery management
- Data integrity management
- Data dictionary management
- Data storage management
- Databasecommunication

- Improved data sharing**:**
  - The DBMS helps create an environment in which end users have better accessto more data and better-managed data.
  - Such access makes it possible for end users to respond quickly to changes intheir environment.

- Improved data security:
  - The more users access the data, the greater the risks of data security breaks.
  - A DBMS provides a framework for better enforcement of data privacy and securitypolicies.

- Better data integration:
  - It provides view of the organizations operations and a clearer view of the bigpicture.
  - It becomes much easier to see how actions in one segment of the companyaffect other segment (part).
  - Minimized data inconsistency:
  - Data inconsistency exists when different versions of the same data appear in differentplaces.
  - The probability of data inconsistency is greatly reduced in properly designeddatabase.

- Improve data access:
  - The DBMS makes it possible to produce quick answers to ad-hoc queries from adatabase
  - A query is a specific request issued to the DBMS for data manipulation.
  - The DBMS sends back an answer to the application.

- Improved decision making:
  - Better managed data and improved data access create
  - It possible to generate better quality information
  - That information is used to take better decisions.

- Increased end-user productivity:
  - The availability of data, combined with the tools
  - That transforms data into usable information.
  - The end users to take quick decisions
  - That is used to success and failure in the global.

- Multi User Access Control
  - DBMS provide many users can access data concurrently with out compromisingthe integrity of the database
  - It can control to access multi user
  - It uses to multiple users can access the database.

- Backup and Recovery Management
  - DBMS provides backup and data recovery
  - It generates data safety and integrity.
  - The special utility that allow the DBA to perform routine and special backup
  - And restore procedures when power failure or database failure.

- Data Integrity Management
  - DBMS supports and implements integrity rules
  - Like not null, check, unique, default, primary key, and foreign key.

- Data dictionary management:
  - The DBMS requires that definitions of the data elements and their relationship(metadata) be stored in a data dictionary.
  - Any change made in a database structures are automatically recorded in the datadictionary,
  - There supports to modify all the programs that access the changed structure.

- Data storage management:
  - The DBMS creates the complex (parts) structures required for data storage
  - A modern DBMS System provides storage not only for the data.
  - But also for related data entry forms or screen definitions, report definitions, data validations rules, procedural code, structures to handle video and picture formats andso on.

- Database communication interfaces:
  - Current generation DBMS provide special communications
  - It allows the database to accept end user requests within a computer networkenvironment.
  - In fact, database communications capabilities are an essential feature of the modernDBMS

Disadvantages of DBMS

Database systems main disadvantages are

1. Increased costs
2. Management complexity
3. Maintaining currency
4. Vendor dependence
5. Frequent upgrade/replacement cycles

1. Increased costs:

a. Database System required sophisticated hardware and software and highly skilledpersonnel.
b. The cost of maintain the hardware, software and personnel required to operateand manage a database system can be large.

2. Management Complexity:
   a. Database Systems interface with many different technologies
   b. And have a significant impact on a company's resources and culture.
   c. Database systems hold crucial company data
   d. That are accessed from multiple sources, security issues must be assessedconstantly.

3. Maintaining currency:
   a. To maximize the efficiency of the database system, you must key your systemcurrent.
   b. User must perform frequent updates and apply.
   c. The latest patches and security measures to all components.
   d. Because database technologies advances rapidly, personnel training cost tend tobe significant.

4. Vendor dependence:
Given the heavy investment in technology and personnel training, companiesmight be reluctant to change database vendors.

5. Frequent upgrade / replacement cycles:
   a. DBMS vendors (seller) frequently upgrade their products by adding newfunctionality.
   b. Such new features often the software. Some of these versions require hardwareupgrades.

VIEW OF DATA -- DATA ABSTRACTION, INSTANCES AND SCHEMAS

Abstraction is one of the main features of database systems. Hiding irrelevant details from user and providing abstract view of data to users, helps in easy and efficient **user-database** interaction. The view level provides the "**view of data**" to the users and hides the irrelevant details such as data relationship, database schema, constraints, security etc from the user.

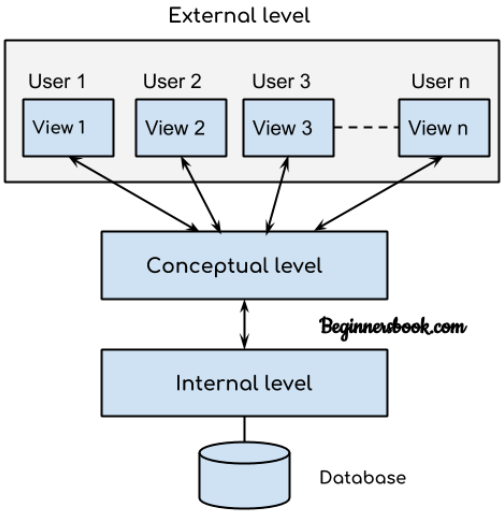
1. **Data abstraction:**Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction.
2. **Instance and schema**: Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema. 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.

DATA ABSTRACTION

Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called **data abstraction**. The term "irrelevant" used here with respect to the user, it doesn't mean that the hidden data is not relevant with regard to the whole database. It just means that the **user is not concerned about that data**.

For example: When you are booking a train ticket, you are not concerned how data is processing at the back end when you click "book ticket", what processes are happening when you are doing online payments. **You are just concerned about the message that pops up when your ticket is successfully booked**. This doesn't mean that the process happening at the back end is not relevant, it just means that you as a user are not concerned what is happening in the database.

Three levels of abstraction



This architecture has three levels:
1. External level
2. Conceptual level
3. Internal level

1. External level

It is also called **view level**. The reason this level is called "view" is because several users can view their desired data from this level which is internally fetched from database with the help of conceptual and internal level mapping.

The user doesn't need to know the database schema details such as data structure, table definition etc. user is only concerned about data which is what returned back to the view level after it has been fetched from database (present at the internal level).

External level is the "**top level**" of the Three Level DBMS Architecture.

2. Conceptual level

It is also called **logical level**. The whole design of the database such as relationship among data, schema of data etc. are described in this level.

Database constraints and security are also implemented in this level of architecture. This level is maintained by DBA (database administrator).

3. Internal level

This level is also known as physical level. This level describes how the data is actually stored in the storage devices. This level is also responsible for allocating space to the data. This is the lowest level of the architecture.

Example: Let's say we are storing customer information in a customer table. At **physical level** these records can be described as blocks of storage (bytes, gigabytes, terabytes etc.) in memory. These details are often hidden from the programmers.

At the **logical level** these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems.

At **view level**, user just interact with system with the help of GUI and enter the details at the screen, they are not aware of how the data is stored and what data is stored; such details are hidden from them.

DBMS Schema

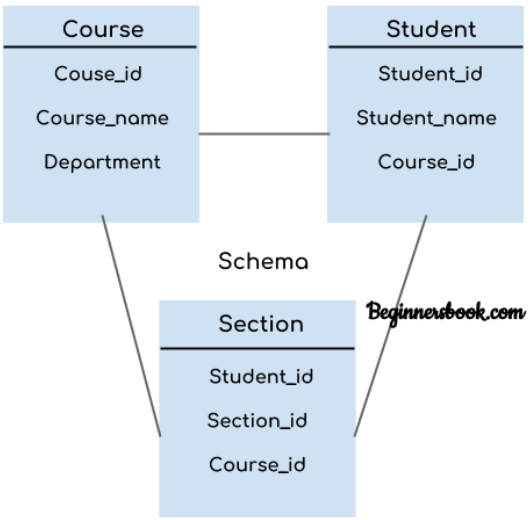
Definition of schema: Design of a database is called the schema. For example: An **employee** table in database exists with the following attributes:

```
EMP_NAME  EMP_ID   EMP_ADDRESS  EMP_CONTACT
--------          ------          -----------         -----------
```

This is the schema of the **employee** table. Schema defines the attributes of tables in the database. **Schema is of three types: Physical schema, logical schema and view schema.**

- Schema represents the **logical view** of the database. It helps you understand what data needs to go where.
- Schema can be represented by a diagram as shown below.
- Schema **helps the database users to understand the relationship between data**. This helps in efficiently performing operations on database such as insert, update, delete, search etc.

In the following diagram, 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 attributes in tables that belong to a particular database. The value of these attributes at a moment of time is called the instance of that database.

For example, we have seen the schema of table "employee" above. Let's see the table with the data now. At this moment the table contains two rows (records). This is the the current instance of the table "employee" because this is the data that is stored in this table at this particular moment of time.

EMP_NAME	EMP_ID	EMP_ADDRESS	EMP_CONTACT
Chaitanya	101	Noida	95********
Ajeet	102	Delhi	99********

Let's take another example: Let's 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. 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, this changes over time as and when we add, delete or update data in the database.

Data models

Data modeling, the first step in designing a database, refers to the process of creating a specific data model for a definite problem domain (area or server).

A data model is a relatively simple representation, usually graphical, of more complex real word data structures.
A model is an abstraction (concept) of a more complex real-word object or event. The database environment, a data model represents data structures and their characteristics, relations, constraints, transformations and other constraints with the purpose of supporting a specific problem domain.

THE IMPORTANCE OF DATA MODELS: Data models can facilitate interaction among the designer the application programmer, and the end user. A well developed data model can even foster improved understanding of the organization for which the database design is developed.

- **THE EVOLUTION OF DATA MODELS**:

Generation	Time	Model	Examples	Comments
First	1960s 1970s	File System	VMS/VSA,	Used mainly on IBM mainFrame Systems. Managed records, not Relationships
Second	1970s	Hierarchical and Network Data Model	IMS ADABAS IDS-II	Early data base system Navigational All Accesses
Third	Mid 1970s to present	Relational Data Model	DB2 Oracle MSSQL – Server MySQL	Conceptual simplicity Entity Relationship modelingand support for Relational Data Modeling

Fourth	Mid 1980s to Present	Object Oriented Extended Relational	Versant FastObjects.Net Objectivity/DB DB/2 UDB Oracle 10g	Support complex data Extended relational products support object and data warehousing.
Next Generation	Present Future	XML	db XML Tamino DB2/UDB Oracle 10g MS SQL Server	Organization and Management of and structured data. Relational and Object modelsadd supports for XML documents.

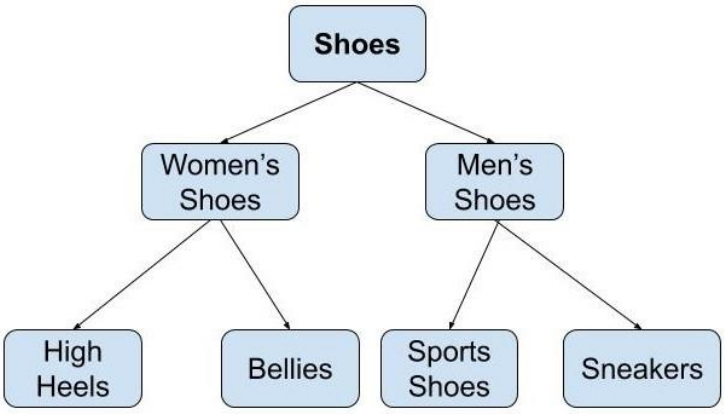
Types of Data Model

Data Model gives us an idea that how the final system will look like after its complete implementation. It defines the data elements and the relationships between the data elements. Data Models are used to show how data is stored, connected, accessed and updated in the database management system. Here, we use a set of symbols and text to represent the information so that members of the organisation can communicate and understand it. Though there are many data models being used nowadays but the Relational model is the most widely used model. Apart from the Relational model, there are many other types of data models about which we will study in details in this blog. Some of the Data Models in DBMS are:

1. Hierarchical Model
2. Network Model
3. Entity-Relationship Model
4. Relational Model
5. Object-Oriented Data Model
6. Object-Relational Data Model
7. Flat Data Model
8. Semi-Structured Data Model
9. Associative Data Model
10. Context Data Model

Hierarchical Model

Hierarchical Model was the first DBMS model. This model organises the data in the hierarchical tree structure. The hierarchy starts from the root which has root data and then it expands in the form of a tree adding child node to the parent node. This model easily represents some of the real-world relationships like food recipes, sitemap of a website etc. ***Example:*** We can represent the relationship between the shoes present on a shopping website in the following way:



Hierarchical Model

Features of a Hierarchical Model

1. ***One-to-many relationship:*** The data here is organised in a tree-like structure where the one-to-many relationship is between the datatypes. Also, there can be only one path from parent to any node. ***Example:*** In the above example, if we want to go to the node *sneakers* we only have one path to reach there i.e through men's shoes node.
2. ***Parent-Child Relationship:*** Each child node has a parent node but a parent node can have more than one child node. Multiple parents are not allowed.
3. ***Deletion Problem:*** If a parent node is deleted then the child node is automatically deleted.
4. ***Pointers:*** Pointers are used to link the parent node with the child node and are used to navigate between the stored data. *Example:* In the above example the '*shoes*' node points to the two other nodes '*women shoes*' node and '*men's shoes*' node.

Advantages of Hierarchical Model

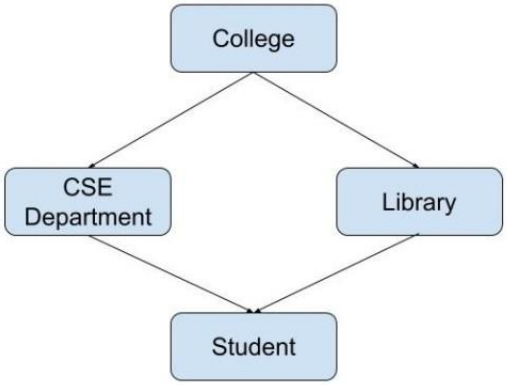
- It is very simple and fast to traverse through a tree-like structure.
- Any change in the parent node is automatically reflected in the child node so, the integrity of data is maintained.

Disadvantages of Hierarchical Model

- Complex relationships are not supported.
- As it does not support more than one parent of the child node so if we have some complex relationship where a child node needs to have two parent node then that can't be represented using this model.
- If a parent node is deleted then the child node is automatically deleted.

Network Model

This model is an extension of the hierarchical model. It was the most popular model before the relational model. This model is the same as the hierarchical model, the only difference is that a record can have more than one parent. It replaces the hierarchical tree with a graph. ***Example:*** In the example below we can see that node student has two parents i.e. CSE Department and Library. This was earlier not possible in the hierarchical model.



Network Model

Features of a Network Model

1. ***Ability to Merge more Relationships:*** In this model, as there are more relationships so data is more related. This model has the ability to manage one-to-one relationships as well as many-to-many relationships.
2. ***Many paths:*** As there are more relationships so there can be more than one path to the same record. This makes data access fast and simple.
3. ***Circular Linked List:*** The operations on the network model are done with the help of the circular linked list. The current position is maintained with the help of a program and this position navigates through the records according to the relationship.

Advantages of Network Model

- The data can be accessed faster as compared to the hierarchical model. This is because the data is more related in the network model and there can be more than one path to reach a particular node. So the data can be accessed in many ways.
- As there is a parent-child relationship so data integrity is present. Any change in parent record is reflected in the child record.

Disadvantages of Network Model

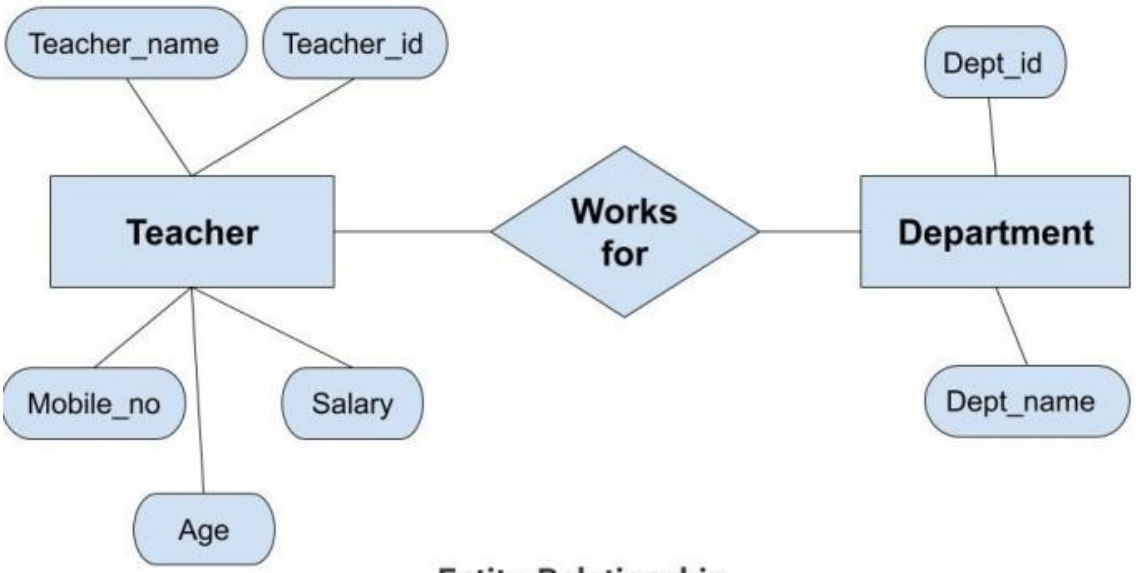
- As more and more relationships need to be handled the system might get complex. So, a user must be having detailed knowledge of the model to work with the model.
- Any change like updation, deletion, insertion is very complex.

Entity-Relationship Model

Entity-Relationship Model or simply ER Model is a high-level data model diagram. In this model, we represent the real-world problem in the pictorial form to make it easy for the stakeholders to understand. It is also very easy for the developers to understand the system by just looking at the ER diagram. We use the ER diagram as a visual tool to represent an ER Model. ER diagram has the following three components:

- ***Entities:*** Entity is a real-world thing. It can be a person, place, or even a concept. *Example:* Teachers, Students, Course, Building, Department, etc are some of the entities of a School Management System.
- ***Attributes:*** An entity contains a real-world property called attribute. This is the characteristics of that attribute. *Example:* The entity teacher has the property like teacher id, salary, age, etc.
- ***Relationship:*** Relationship tells how two attributes are related. *Example:* Teacher works for a department.

Example:



Entity-Relationship Model

In the above diagram, the entities are Teacher and Department. The attributes of ***Teacher*** entity are Teacher_Name, Teacher_id, Age, Salary, Mobile_Number. The attributes of entity ***Department*** entity are Dept_id, Dept_name. The two entities are connected using the relationship. Here, each teacher works for a department.

Features of ER Model

- ***Graphical Representation for Better Understanding:*** It is very easy and simple to understand so it can be used by the developers to communicate with the stakeholders.
- ***ER Diagram:*** ER diagram is used as a visual tool for representing the model.
- ***Database Design:*** This model helps the database designers to build the database and is widely used in database design.

Advantages of ER Model

- ***Simple:*** Conceptually ER Model is very easy to build. If we know the relationship between the attributes and the entities we can easily build the ER Diagram for the model.
- ***Effective Communication Tool***: This model is used widely by the database designers for communicating their ideas.
- ***Easy Conversion to any Model***: This model maps well to the relational model and can be easily converted relational model by converting the ER model to the table. This model can also be converted to any other model like network model, hierarchical model etc.

Disadvatages of ER Model

- ***No industry standard for notation:*** There is no industry standard for developing an ER model. So one developer might use notations which are not understood by other developers.
- ***Hidden information:*** Some information might be lost or hidden in the ER model. As it is a high-level view so there are chances that some details of information might be hidden.

Relational Model

Relational Model is the most widely used model. In this model, the data is maintained in the form of a two-dimensional table. All the information is stored in the form of row and columns. The basic structure of a relational model is tables. So, the tables are also called *relations* in the relational model. ***Example:*** In this example, we have an Employee table.

Emp_id	Emp_name	Job_name	Salary	Mobile_no	Dep_id	Project_id
AfterA001	John	Engineer	100000	9111037890	2	99
AfterA002	Adam	Analyst	50000	9587569214	3	100
AfterA003	Kande	Manager	890000	7895212355	2	65

EMPLOYEE TABLE

Features of Relational Model

- ***Tuples***: Each row in the table is called tuple. A row contains all the information about any instance of the object. In the above example, each row has all the information about any specific individual like the first row has information about John.
- ***Attribute or field:*** Attributes are the property which defines the table or relation. The values of the attribute should be from the same domain. In the above example, we have different attributes of the *employee* like Salary, Mobile_no, etc.

Advnatages of Relational Model

- ***Simple:*** This model is more simple as compared to the network and hierarchical model.
- ***Scalable:*** This model can be easily scaled as we can add as many rows and columns we want.
- ***Structural Independence:*** We can make changes in database structure without changing the way to access the data. When we can make changes to the database structure without affecting the capability to DBMS to access the data we can say that structural independence has been achieved.

Disadvantages of Relatinal Model

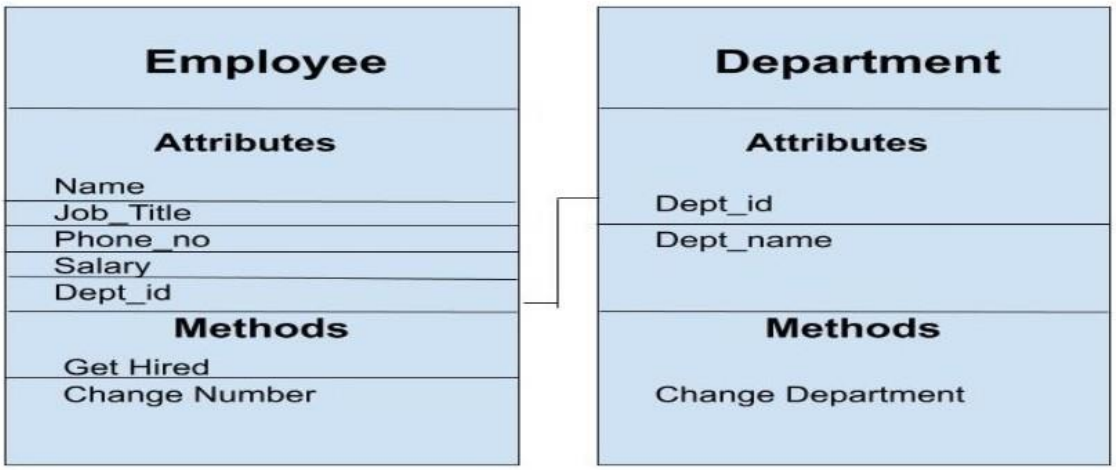
- ***Hardware Overheads:*** For hiding the complexities and making things easier for the user this model requires more powerful hardware computers and data storage devices.
- ***Bad Design:*** As the relational model is very easy to design and use. So the users don't need to know how the data is stored in order to access it. This ease of design can lead to the development of a poor database which would slow down if the database grows.

But all these disadvantages are minor as compared to the advantages of the relational model. These problems can be avoided with the help of proper implementation and organisation.

Object-Oriented Data Model

The real-world problems are more closely represented through the object-oriented data model. In this model, both the data and relationship are present in a single structure known as an object. We can store audio, video, images, etc in the database which was not possible in the relational model(although you can store audio and video in

relational database, it is adviced not to store in the relational database). In this model, two are more objects are connected through links. We use this link to relate one object to other objects. This can be understood by the example given below.



Object_Oriented_Model

In the above example, we have two objects Employee and Department. All the data and relationships of each object are contained as a single unit. The attributes like Name, Job_title of the employee and the methods which will be performed by that object are stored as a single object. The two objects are connected through a common attribute i.e the Department_id and the communication between these two will be done with the help of this common id.

Object-Relational Model

As the name suggests it is a combination of both the relational model and the object-oriented model. This model was built to fill the gap between object-oriented model and the relational model. We can have many advanced features like we can make complex data types according to our requirements using the existing data types. The problem with this model is that this can get complex and difficult to handle. So, proper understanding of this model is required.

Flat Data Model

It is a simple model in which the database is represented as a table consisting of rows and columns. To access any data, the computer has to read the entire table. This makes the modes slow and inefficient.

Semi-Structured Model

Semi-structured model is an evolved form of the relational model. We cannot differentiate between data and schema in this model. ***Example:*** Web-Based data sources which we can't differentiate between the schema and data of the website. In this model, some entities may have missing attributes while others may have an extra attribute. This model gives flexibility in storing the data. It also gives flexibility to the attributes. ***Example:*** If we are storing any value in any attribute then that value can be either atomic value or a collection of values.

Associative Data Model

Associative Data Model is a model in which the data is divided into two parts. Everything which has independent existence is called as an *entity* and the relationship among these entities are called *association*. The data divided into two parts are called items and links.

- ***Item***: Items contain the name and the identifier(some numeric value).

- ***Links:*** Links contain the identifier, source, verb and subject.

Example: Let us say we have a statement "The world cup is being hosted by London from 30 May 2020". In this data two links need to be stored:

1. The world cup is being hosted by London. The source here is 'the world cup', the verb 'is being' and the target is 'London'.
2. ...from 30 May 2020. The source here is the previous link, the verb is 'from' and the target is '30 May 2020'.

This is represented using the table as follows:

Items

Identifiers	Name
89	The world cup
95	Is being hosted
40	By London
44	from
10	30 May 2020

Links

Identifiers	Source	Verb	Target
70	89	95	40
75	70	44	10

ASSOCIATIVE MODEL

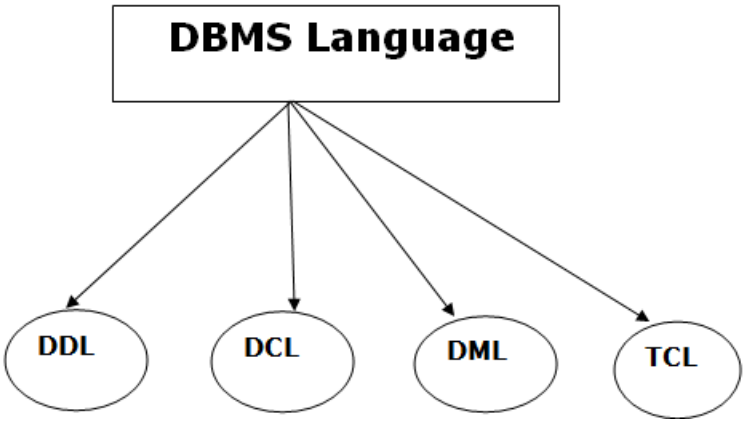
Context Data Model

Context Data Model is a collection of several models. This consists of models like network model, relational models etc. Using this model we can do various types of tasks which are not possible using any model alone.

DATABASE LANGUAGES -- DATA DEFINITION LANGUAGE, DATA MANIPULATION LANGUAGE

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:



1. Data Definition Language

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

2. Data Manipulation Language

DML stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data.
- **Lock Table:** It controls concurrency.

3. Data Control Language

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.

  (But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

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

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

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

DATABASE ARCHITECTURE

The architecture of DBMS depends on the computer system on which it runs. For example, in a client-server DBMS architecture, the database systems at server machine can run several requests made by client machine. We will understand this communication with the help of diagrams.

Types of DBMS Architecture : There are three types of DBMS architecture:

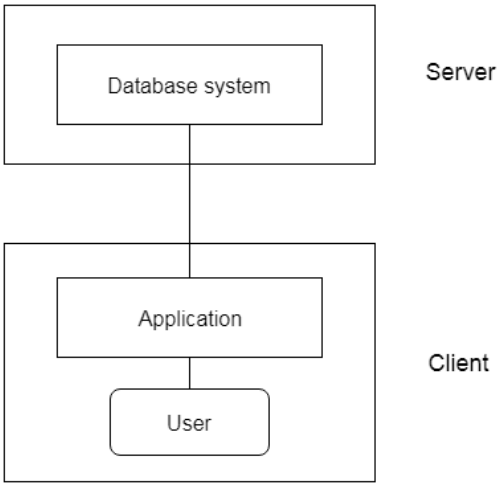
1. Single tier architecture
2. Two tier architecture
3. Three tier architecture

1. Single tier architecture

In this type of architecture, the database is readily available on the client machine, any request made by client doesn't require a network connection to perform the action on the database.

For example, lets say you want to fetch the records of employee from the database and the database is available on your computer system, so the request to fetch employee details will be done by your computer and the records will be fetched from the database by your computer as well. This type of system is generally referred as local database system

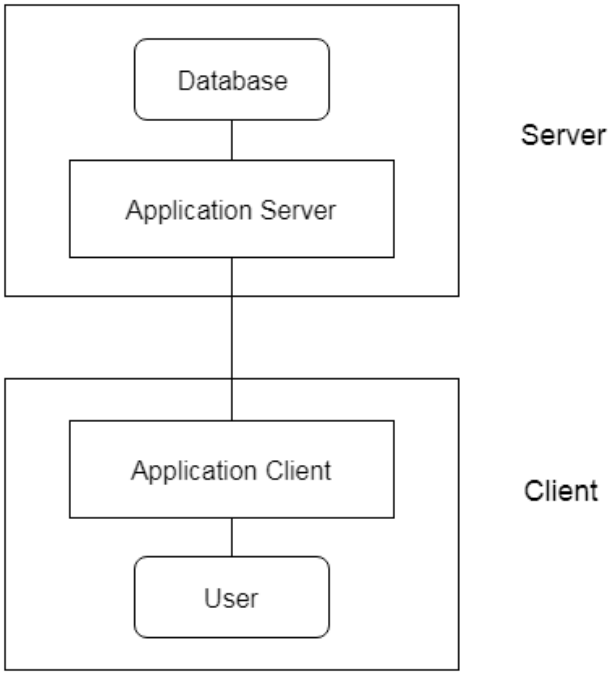
2. Two tier architecture



In two-tier architecture, the Database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a reliable network as shown in the above diagram.

Whenever client machine makes a request to access the database present at server using a query language like sql, the server perform the request on the database and returns the result back to the client. The application connection interface such as JDBC, ODBC are used for the interaction between server and client.

3. Three tier architecture

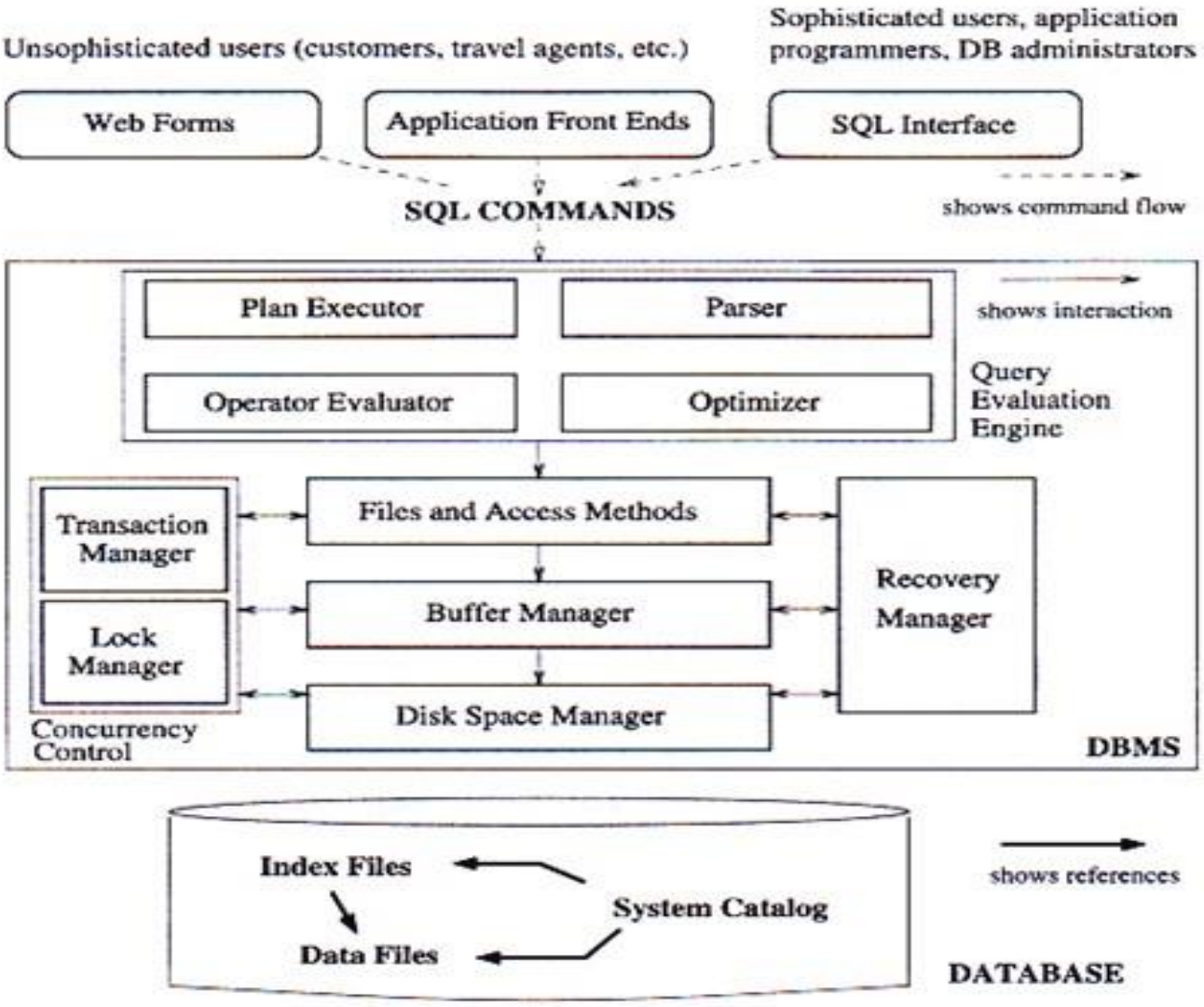


In three-tier architecture, another layer is present between the client machine and server machine. In this architecture, the client application doesn't communicate directly with the database systems present at the server machine, rather the client application communicates with server application and the server application internally communicates with the database system present at the server

Structure of DBMS:

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the **storage manager** and the **query processor** components. The storage manager is important because databases typically require a large amount of storage space. The query processor is important because it helps the database system simplify and facilitate access to data.

It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.



Database Users:

Users are differentiated by the way they expect to interact with the system:

- **Application programmers:**
    - Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces.
    - Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports without writing a program.
- **Sophisticated users:**
    - Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language.
    - They submit each such query to a query processor, whose function is to break down DML statements into instructions that the storage manager understands.
- **Specialized users :**
    - Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.
    - Among these applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

- **Naïve users :**
    - Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.
    - For example, a bank teller who needs to transfer $50 from account A to account B invokes a program called transfer. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred.

Database Administrator:

- Coordinates all the activities of the database system. The database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties include:
    - **Schema definition:** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
    - **Storage structure and access method definition.**
    - **Schema and physical organization modification:** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.
    - **Granting user authority to access the database:** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access.
    - **Specifying integrity constraints.**
    - **Monitoring performance and responding to changes in requirements.**

Query Processor

The query processor components include

· **DDL interpreter,** which interprets DDL statements and records the definitions in the data dictionary.

· **DML compiler,** which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.

A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs **query optimization**, that is, it picks the lowest cost evaluation plan from among the alternatives.

· **Query evaluation engine,** which executes low-level instructions generated by the DML compiler.

Storage Manager

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

The storage manager components include:

· **Authorization and integrity manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

· **Transaction manager**, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.

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

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

Transaction Manager

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

- **Data structures implemented by storage manager.**
- **Data files:** Stored in the database itself.
- **Data dictionary:** Stores metadata about the structure of the database.
- **Indices:** Provide fast access to data items.

DATABASE USERS AND ADMINISTRATORS

Database Users

Database users are the ones who really use and take the benefits of the database. There will be different types of users depending on their needs and way of accessing the database.

1. **Application Programmers –** They are the developers who interact with the database by means of DML queries. These DML queries are written in the application programs like C, C++, JAVA, Pascal, etc. These queries are converted into object code to communicate with the database. For example, writing a C program to generate the report of employees who are working in a particular department will involve a query to fetch the data from the database. It will include an embedded SQL query in the C Program.
2. **Sophisticated Users –** They are database developers, who write SQL queries to select/insert/delete/update data. They do not use any application or programs to request the database. They directly interact with the database by means of a query language like SQL. These users will be scientists, engineers, analysts who thoroughly study SQL and DBMS to apply the concepts in their requirements. In short, we can say this category includes designers and developers of DBMS and SQL.
3. **Specialized Users –** These are also sophisticated users, but they write special database application programs. They are the developers who develop the complex programs to the requirement.
4. **Stand-alone Users –** These users will have a stand-alone database for their personal use. These kinds of the database will have readymade database packages which will have menus and graphical interfaces.
5. **Native Users –** these are the users who use the existing application to interact with the database. For example, online library system, ticket booking systems, ATMs etc which has existing application and users use them to interact with the database to fulfill their requests.

Database Administrators

The life cycle of a database starts from designing, implementing to the administration of it. A database for any kind of requirement needs to be designed perfectly so that it should work without any issues. Once all the design is complete, it needs to be installed. Once this step is complete, users start using the database. The database grows as

the data grows in the database. When the database becomes huge, its performance comes down. Also accessing the data from the database becomes a challenge. There will be unused memory in the database, making the memory inevitably huge. This administration and maintenance of the database are taken care of by the database Administrator – DBA.

A DBA has many responsibilities. A good-performing database is in the hands of DBA.

- **Installing and upgrading the DBMS Servers: –** DBA is responsible for installing a new DBMS server for the new projects. He is also responsible for upgrading these servers as there are new versions that come into the market or requirement. If there is any failure in the up-gradation of the existing servers, he should be able to revert the new changes back to the older version, thus maintaining the DBMS working. He is also responsible for updating the service packs/ hotfixes/ patches to the DBMS servers.
- **Design and implementation: –** Designing the database and implementing is also DBA's responsibility. He should be able to decide on proper memory management, file organizations, error handling, log maintenance, etc for the database.
- **Performance tuning: –** Since the database is huge and it will have lots of tables, data, constraints, and indices, there will be variations in the performance from time to time. Also, because of some designing issues or data growth, the database will not work as expected. It is the responsibility of the DBA to tune the database performance. He is responsible to make sure all the queries and programs work in a fraction of seconds.
- **Migrate database servers: –** Sometimes, users using oracle would like to shift to SQL server or Netezza. It is the responsibility of DBA to make sure that migration happens without any failure, and there is no data loss.
- Backup and Recovery: – Proper backup and recovery programs needs to be developed by DBA and has to be maintained him. This is one of the main responsibilities of DBA. Data/objects should be backed up regularly so that if there is any crash, it should be recovered without much effort and data loss.
- **Security: –** DBA is responsible for creating various database users and roles, and giving them different levels of access rights.
- **Documentation: –** DBA should be properly documenting all his activities so that if he quits or any new DBA comes in, he should be able to understand the database without any effort. He should basically maintain all his installation, backup, recovery, security methods. He should keep various reports about database performance.

In order to perform his entire task, he should have very good command over DBMS.

Types of DBA

There are different kinds of DBA depending on the responsibility that he owns.

- **Administrative DBA –** This DBA is mainly concerned with installing, and maintaining DBMS servers. His prime tasks are installing, backups, recovery, security, replications, memory management, configurations, and tuning. He is mainly responsible for all administrative tasks of a database.
- **Development DBA –** He is responsible for creating queries and procedures for the requirement. Basically, his task is similar to any database developer.
- **Database Architect –** Database architect is responsible for creating and maintaining the users, roles, access rights, tables, views, constraints, and indexes. He is mainly responsible for designing the structure of the database depending on the requirement. These structures will be used by developers and development DBA to code.
- **Data Warehouse DBA –**DBA should be able to maintain the data and procedures from various sources in the data warehouse. These sources can be files, COBOL, or any other programs. Here data and programs will be from different sources. A good DBA should be able to keep the performance and function levels from these sources at the same pace to make the data warehouse work.
- **Application DBA –**He acts like a bridge between the application program and the database. He makes sure all the application program is optimized to interact with the database. He ensures all the activities from installing, upgrading, and patching, maintaining, backup, recovery to executing the records work without any issues.
- **OLAP DBA –** He is responsible for installing and maintaining the database in OLAP systems. He maintains only OLAP databases.

Introduction to Database Design: Database design and ER diagrams, Entities, attributes and entity sets, Relationships and relationship sets, Additional features of ER model, Conceptual Design with ER model.

DATABASE DESIGN AND ER DIAGRAMS

Database Design:

Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems. Properly designed database are easy to maintain, improves data consistency and are cost effective in terms of disk storage space. The database designer decides how the data elements correlate and what data must be stored.
The main objectives of database design in DBMS are to produce logical and physical designs models of the proposed database system.

The logical model concentrates on the data requirements and the data to be stored independent of physical considerations. It does not concern itself with how the data will be stored or where it will be stored physically.

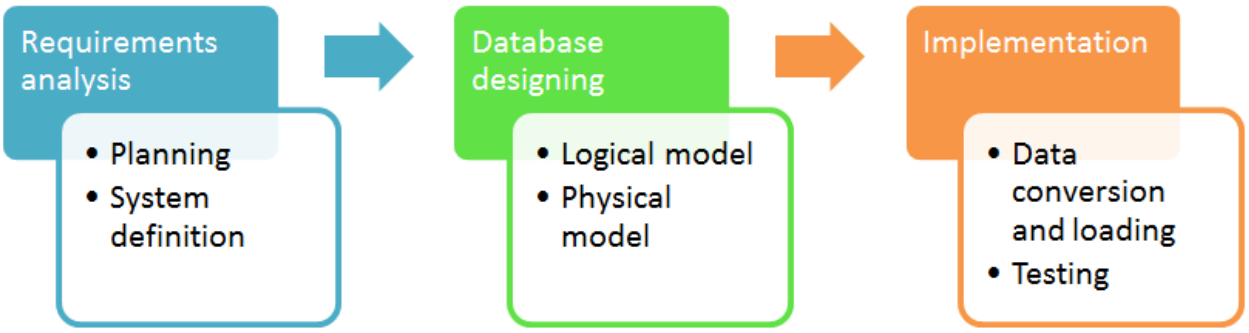
The physical data design model involves translating the logical DB design of the database onto physical media using hardware resources and software systems such as database management systems (DBMS).

It helps produce database systems

1. That meet the requirements of the users
2. Have high performance.

Database design process in DBMS is crucial for **high performance** database system.

Database development life cycle



The database development life cycle has a number of stages that are followed when developing database systems. The steps in the development life cycle do not necessarily have to be followed religiously in a sequential manner. On small database systems, the process of database design is usually very simple and does not involve a lot of steps. In order to fully appreciate the above diagram, let's look at the individual components listed in each step for overview of design process in DBMS.

Requirements analysis

- **Planning** – This stages of database design concepts are concerned with planning of entire Database Development Life Cycle. It takes into consideration the Information Systems strategy of the organization.

- **System definition** – This stage defines the scope and boundaries of the proposed database system.

Database designing

- **Logical model** – This stage is concerned with developing a database model based on requirements. The entire design is on paper without any physical implementations or specific DBMS considerations.
- **Physical model** – This stage implements the logical model of the database taking into account the DBMS and physical implementation factors.

Implementation

- **Data conversion and loading** – this stage of relational databases design is concerned with importing and converting data from the old system into the new database.
- **Testing** – this stage is concerned with the identification of errors in the newly implemented system. It checks the database against requirement specifications.

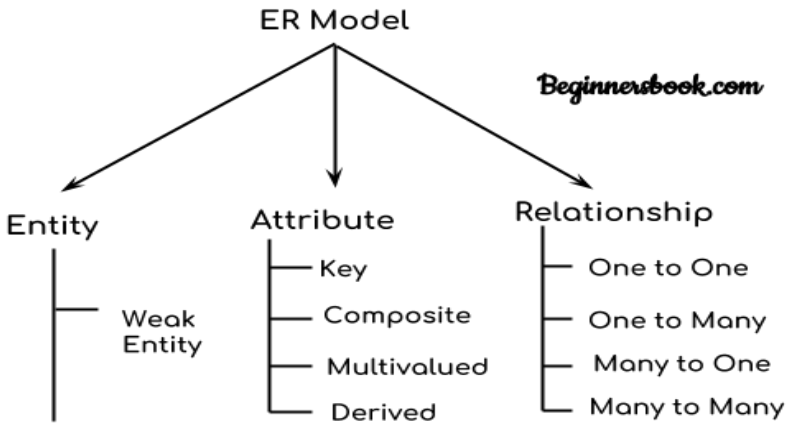
Two Types of Database Techniques

1. **Normalization**
2. **ER Modeling**

ER DIAGRAM

E-R model is a graphical representation of the logical structure of the database. In this model, we represent the real-world problem in the pictorial form to make it easy for the stakeholders to understand. It is also very easy for the developers to understand the system by just looking at the ER diagram. We use the ER diagram as a visual tool to represent an ER Model. ER diagram has the following three components:

Components of an E-R diagram

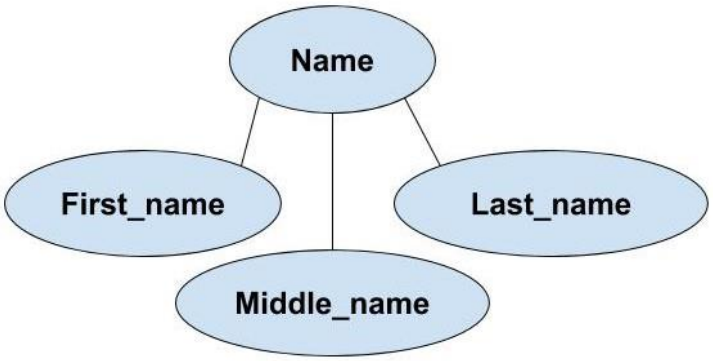


- ***Entities:*** Entity is a real-world thing. It can be a person, place, or even a concept. It is represented by a rectangle. Each entity has one of its attributes(attributes are properties of an entity) as the primary key which can define it uniquely. Such an entity is called a ***strong entity***. There are some entities which cant be uniquely identified from there attributes. Such an entity is called a ***weak entity***. We will study them in details in some other blog. ***Example:*** Teachers, Student, Course, Building, Department, etc are some of the entities of a School Management System. Student entity is represented as below.

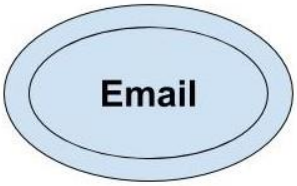
- ***Attributes:*** An entity contains a real-world property called attribute. This is the characteristics of that attribute. It is represented by an oval. ***Example:*** The entity car has properties like Car_no, Color, Price, etc. There are many types of attributes which we will study in detail in some other blog. Here, we will have an overview of it. A simple attribute contains an atomic value which cannot be further divided. ***Example:*** Roll Number of Student.

Key Attribute*: Key attribute is used to uniquley identify the records and it represents the primary key of the table. It is represented by oval and the text in it is underlined. Example: If we have Roll_no as the key attribute then it would be represneted as:*

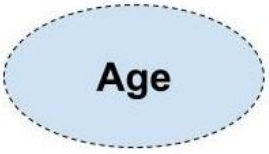
Composite Attribute: *This attribute can be further divided into other attributes.* ***Example:*** *Name of a student can be further divided into first name, middle name and last name.*



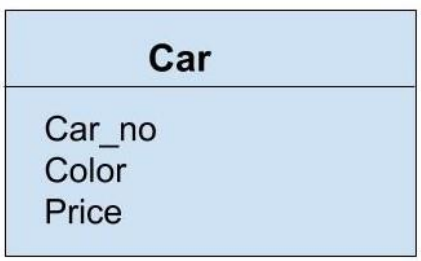
Multivalued Attribute: *This attribute has more than one value. It is represented by a double oval.* ***Example:*** *A student can have more than one e-mail address.*



Derived Attribute: *The value of this attribute are derived from some other attributes. This is done mainly because the value for such attribute keeps on changing. It is represented by a dashed oval.* ***Example:*** *The value of age attribute is derived from the DOB(date of birth) attribute.*

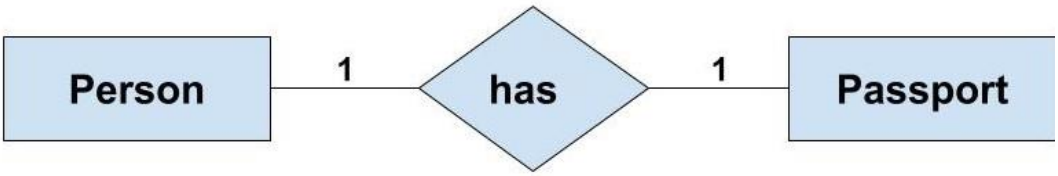


Note: Entities along with attributes are called entity type or schema. ***Example:*** Car entity has attributes Car_no, Color, Price. So, we represent an entity type as follows:

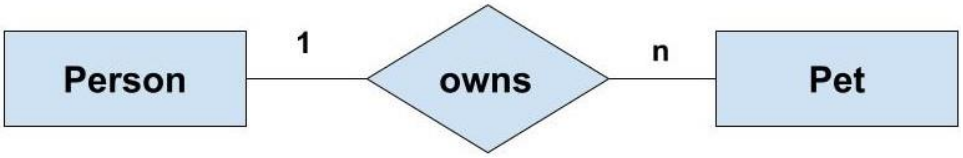


- ***Relationship:*** Association between two entities is called a relationship. Entities take part in the relationship. It is represented by a diamond shape. ***Example:*** Teacher teaches a student. There are three types of relationships that can exist between two entities.

One-to-One Relationship: *Such a relationship exists when each record of one table is related to only one record of the other table.* ***Example:*** *If there are two entities 'Person' and 'Passport'. So, each person can have only one passport and each passport belongs to only one person.*

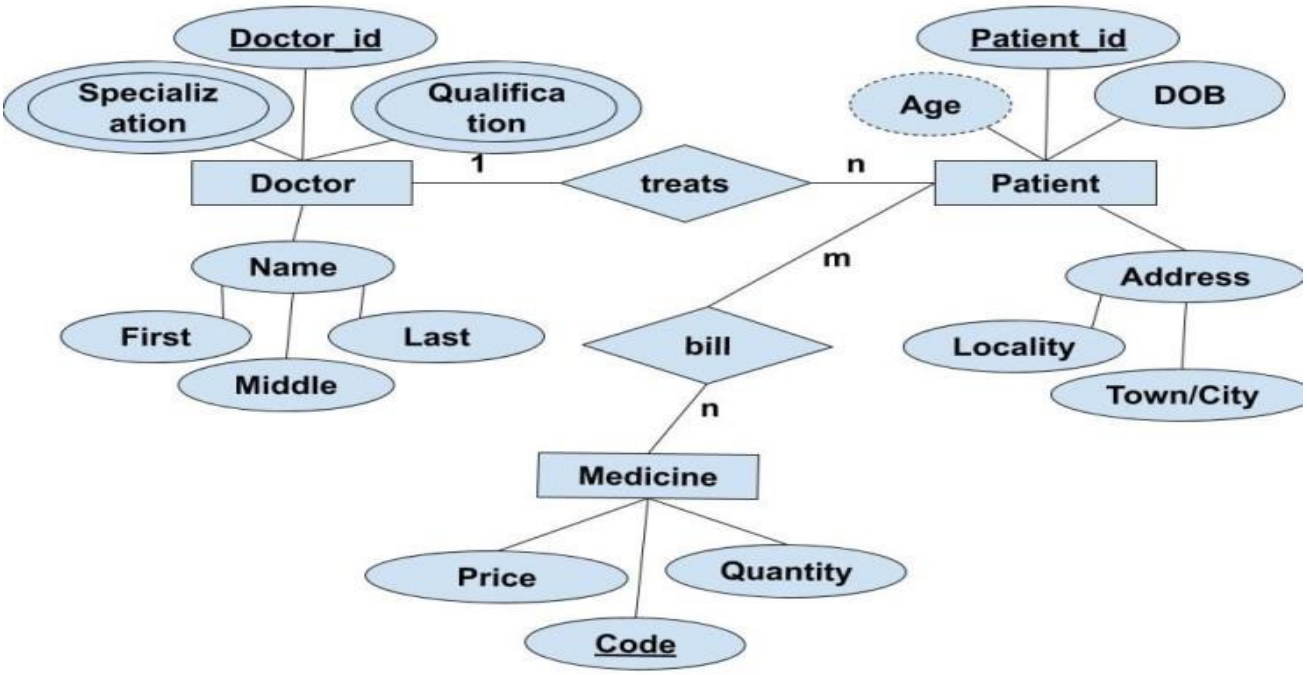


One-to-Many Relationship: *Such a relationship exists when each record of one table can be related to one or more than one record of the other table.* ***Example:*** *If there are two entities 'Owner' and 'Pet' then each owner can have more than one pet but each pet is owned by only one owner.*



Many-to-Many Relationship: *Such a relationship exists when each record of the first table can be related to one or more than one record of the second table and a single record of the second table can be related to one or more than one record of the first table.* ***Example:*** *If there are two entities 'Customer' and 'Product' then each customer can buy more than one product and a product can be bought by many different customers.*

Using the above knowledge of how to make an ER diagram, we can make an ER diagram for a hospital management system. We will have three entities i.e Doctor, Patient, Medicine.



Hospital Management System

In the above diagram, entity Doctor has key attribute 'doctor_id' which will be used to identify the doctors. It also has two multivalued attributes as 'specialization' and 'qualification' as a doctor may have more than one qualification and may be specialized in more than one fields. The Doctor and Patient entity have a one-to-many relationship as a Doctor may treat more than one patient. Similarly, Patient and Medicine have a many-to-many relationship as a patient may buy more than one medicine and vice-versa. 'Code' is the key attribute for Medicine which is unique for every medicine. The Patient has many attributes Patient_id, DOB, Age, etc. 'Age' is the derived attribute here. Also, it has a composite attribute 'Address' which can further be divided into two attributes 'Locality' and 'Town'.

Features of ER Model

- ***Graphical Representation for Better Understanding:*** It is very easy and simple to understand so it can be used by the developers to communicate with the stakeholders.
- ***ER Diagram:*** ER diagram is used as a visual tool for representing the model. As seen above the ER diagram makes it very easy for us to represent the real-world problems.
- ***Database Design:*** This model helps the database designers to build the database and is widely used in database design. ER model as discussed above provides a base for designing the database. ***Example:*** In the above hospital management system, we can create the relational database according to the attributes, entities and the relationship among them. We can have a separate table for each entity. The attributes can make the column of each table and the association between the tables is defined according to the relationship according to them.

Advantages of ER Model

- ***Simple:*** Conceptually ER Model is very easy to build. If we know the relationship between the attributes and the entities we can easily build the ER Diagram for the model.
- ***Effective Communication Tool***: This model is used widely by the database designers for communicating their ideas.
- ***Easy Conversion to any Model***: This model maps well to the relational model and can be easily converted relational model by converting the ER model to the table. This model can also be converted to any other model like network model, hierarchical model etc.

Disadvantages of ER Model

- ***No industry standard for notation:*** There is no industry standard for developing an ER model. So one developer might use notations which are not understood by other developers.
- ***Hidden information:*** Some information might be lost or hidden in the ER model. As it is a high-level view so there are chances that some details of information might be hidden.

ENTITY, ENTITY SET AND ATTRIBUTE

⇨ ENTITIES:

1. An entity is an object of interest to the end user.
2. An entity is represented by a rectangle containing the entity's name.
3. The entity name, a noun, is usually written in all capital letters.
4. An entity is anything a place, a person, a thing, an event about which data arecollected and store.
5. A collection of entities known as entity set or entity type.
6. Each row in the relational table is known as an entity instance or entity occurrence(event).
7. An entity can be of two types:

Tangible Entity: Tangible Entities are those entities which exist in the real world physically. ***Example:*** Person, car, etc.

Intangible Entity: Intangible Entities are those entities which exist only logically and have no physical existence. ***Example:*** Bank Account, etc.

Ex: If we have a table of a Student (Roll_no, Student_name, Age, Mobile_no) then each student in that table is an entity and can be uniquely identified by their Roll Number i.e Roll_no.

Student

Roll_no	Student_name	Age	Mobile_no
1	Andrew	18	7089117222
2	Angel	19	8709054568
3	Priya	20	9864257315
4	Analisa	21	9847852156

→ **Entity**

ENTITY TYPE

The entity type is a collection of the entity having similar attributes. In the above Student table example, we have each row as an entity and they are having common attributes i.e each row has its own value for attributes Roll_no, Age, Student_name and Mobile_no. So, we can define the above STUDENT table as an entity type because it is a collection of entities having the same attributes. **So, an entity type in an ER diagram is defined by a name(here, STUDENT) and a set of attributes(here, Roll_no, Student_name, Age, Mobile_no).** The table below shows how the data of different entities(different students) are stored.

Student → **Entity Type**

Roll_no	Student_name	Age	Mobile_no
1	Andrew	18	7089117222
2	Angel	19	8709054568
3	Priya	20	9864257315
4	Analisa	21	9847852156

→ **Entity**

The E-R representation of the above Student Entity Type is done below.

There are different types of entities

1. Strong entity.
2. Weak entity.
3. Associative entity (or) Composite entity (or) bridge entity

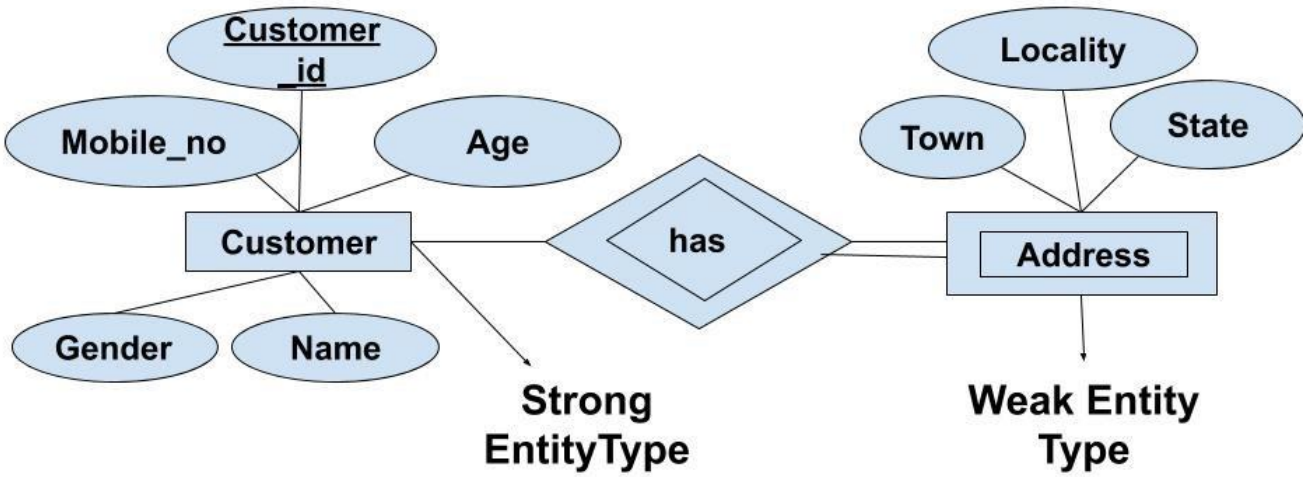
Strong Entities

1. A strong entity type is an entity that exists independently of other entity types.
2. Strong entity type always has a unique characteristic called an identifier.
3. That is an attribute or combination of attribute
4. That uniquely identified each occurrence (incidence) of that entity type.
5. It can be denoted as single line rectangle.

Weak Entities

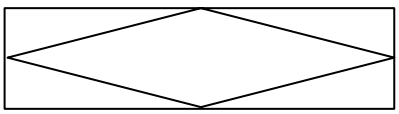
1. A weak entity is an entity type whose existence depends on some other entitytype.
2. The entity has not primary key
3. That is partially or totally derived from the parent entity in the relationship.
4. It is represented by double line rectangle.

Example: *If we have two tables of Customer(Customer_id, Name, Mobile_no, Age, Gender) and Address(Locality, Town, State, Customer_id). Here we cannot identify the address uniquely as there can be many customers from the same locality. So, for this, we need an attribute of Strong Entity Type i.e 'Customer' here to uniquely identify entities of 'Address' Entity Type.*



ASSOCIATIVE ENTITY

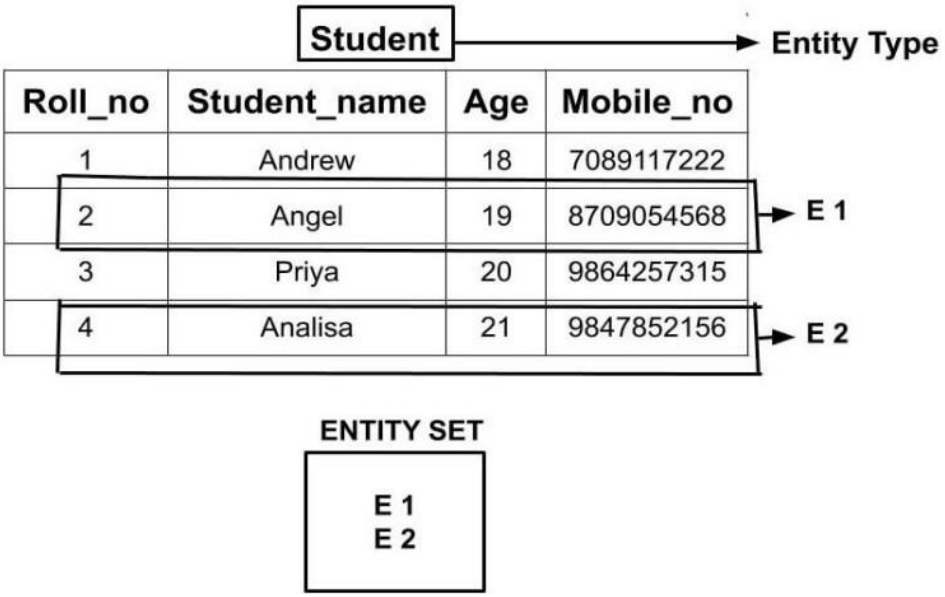
1. An associative entity is an entity type that associates the instances of one ormore entity type.
2. It means a many to many relationship exits
3. It is also called composite entity or bridge entity
4. And it contains attributes that are peculiar to the relationship between those entityinstances.
5. It is represented by a diamond shape with in a rectangle.



ENTITY SET

Entity Set is a collection of entities of the same entity type. In the above example of STUDENT entity type, a collection of entities from the Student entity type would form an entity set. **We can say that entity type is a superset of the entity set as all the entities are included in the entity type.** Let's try to understand this with the help of an example.

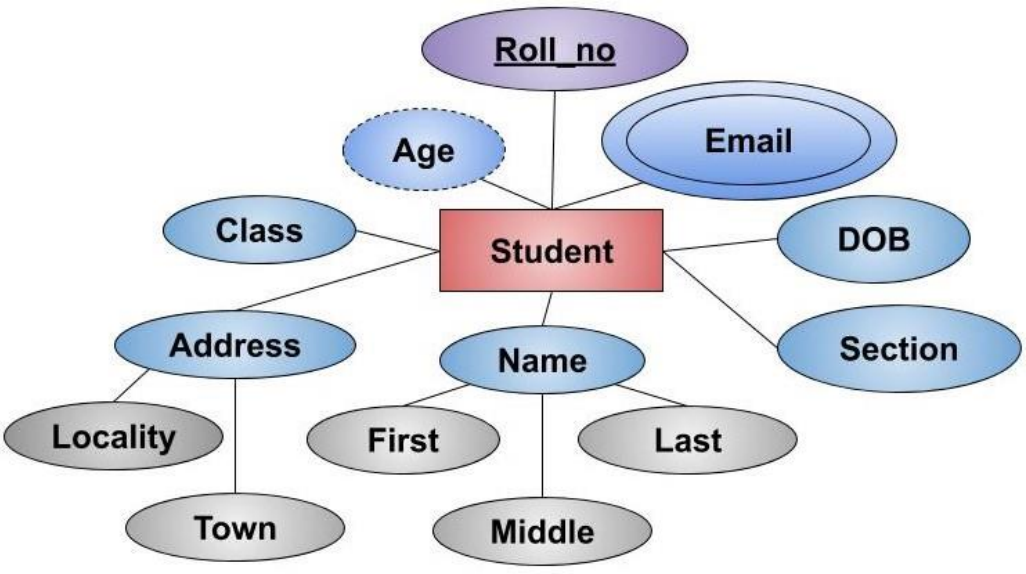
Example 1: *In the below example, two entities E1 (2, Angel, 19, 8709054568) and E2(4, Analisa, 21, 9847852156) form an entity set.*



ATTRIBUTE: **A real-world property of an entity type is called an attribute.** This is the characteristics of an entity. It is represented by an oval or ellipse in E-R diagram. Each attribute can take only a set of permitted values. This is called the ***domain*** of that attribute.

For example, we define the roll_no of the 'Student' by a numeric value. So, the permitted values are only integers and hence, 'integer' is the domain of attribute 'roll_no'. Each attribute is represented by a separate column in a relational table.

The entity 'Student' has properties like Name, Address, Roll_no, Mobile_no, Age, DOB, Class, Section, etc. So, when we make an E-R diagram then Name, Address, Roll_no, Mobile_no, Age, DOB, Section and Class are represented as the attributes of the entity type 'Student'



There are many **types of attributes** which are as follows:

- Simple Attribute & Composite Attribute
- Single Valued Attribute & Multi-valued Attribute
- Stored Attribute & Derived Attribute
- Key Attribute & Non-key Attribute

Simple Attribute & Composite Attribute

This division is made on the basis that if the attribute can be further divided or not into more attributes. If it cannot be further divided it is a simple attribute and if can be further divided it is a composite attribute.

Simple Attribute

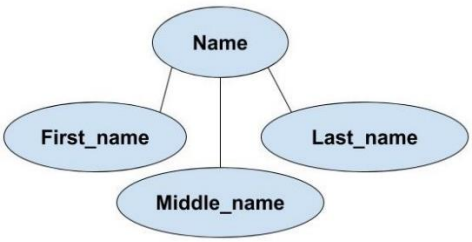
A simple attribute contains an atomic value which cannot be further divided. It is simply represented by an oval. A simple attribute is directly connected to the entity type. While making the E-R diagram, we directly connect the oval with the rectangle.

For example, Roll_no of Student, Age of Student. It is represented in the E-R diagram as:

Composite Attribute

A Composite attribute can be further divided into other attributes. We represent this in E-R diagram by connecting an oval with oval i.e the composite attribute is also represented by oval and the further divided attribute all also represented by ovals. When we convert this E-R diagram to the relational table then we don't have any column of the composite attribute. We have column only for the attribute which came after we further divided the composite attribute.

For example, Name of a student can be further divided into first name, middle name and last name. The composite attribute name is also represented by oval as well as the other attributes are also represented by oval and we connect the all the further divided attributes with the composite attribute. In the table, we will have three columns i.e. First_name, Middle_name, and Last_name. There is no such column called "Name".



Single Valued Attribute & Multi-valued Attribute

This division is made on the basis of how many values can be taken by the attribute. If the attribute can take more than one value, it is a multi-valued attribute. If the attribute takes only one value it is a single-valued attribute.

Single-valued Attribute

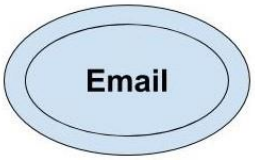
This attribute has only one value. It is represented by a simple oval. Some simple attribute can also be a single-valued attribute. For example, the Section of 'Student' is a simple attribute as it can't be further divided. Also, it is a single-valued attribute because a student can't study in more than one section.

For example, Section of a Student.

Multivalued Attribute

This attribute has more than one value. It is represented by a double oval. Some Composite keys can also be a multivalued attribute. For example, the address attribute of a student can further be divided into 'Locality', 'Town' and 'PIN'. So, we call it a composite attribute. Also, the address of a student can have more than one value. A Student can have more than one Address i.e Permanent Address and Temporary Address. So, it can also be called a multivalued attribute.

For example, A student can have more than one e-mail address.



Stored Attribute & Derived Attribute

This classification is made on the basis that if the attribute is just stored in the database or can be derived from some other attribute.

Stored Attribute

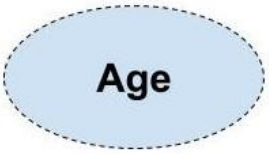
The value of this attribute should be provided by the user. This attribute is stored and can be used for finding the value of other attributes. It cannot be derived from any other attribute. It is also represented by an oval. For Example, The Date of Birth of 'Student' has to be provided by the student.

Example: Date of Birth(DOB) of Student.

Derived Attribute

The value of this attribute is derived from some other attributes. We know the value of some other attribute(stored attribute)and from stored attribute, we are deriving the value of this attribute(derived attribute). This is done mainly because the value for such attribute keeps on changing. It is represented by a dashed oval.

For example, The value of age attribute is derived from the DOB(date of birth) attribute.



Key Attribute & Non-key Attribute

This classification is made on the basis that if the attribute can uniquely identify the entities or not. As the name suggests key attribute will uniquely identify the entities whereas the non-key attributes would not be able to uniquely identify the entities.

Key Attribute

A key attribute is used to uniquely identify the entities of an entity type. In a relational table, it represents the primary key of the table. It is represented by oval and the text in it is underlined. Even if all the other attributes of an entity are the same but the key attribute will always be different.

Example: We have Roll_no as the key attribute of the 'Student' because two students can never has same roll number.

Non-Key Attribute

All the other attributes other than the key attribute are the non-key attributes. Two or more entities can have the same value for this attribute. For example, the Class attribute would have the same value for all those students who are studying in the same class.

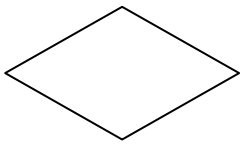
Example: Class, Section, Age, Name etc, are the non-key attributes.

Note: *The same attribute can be of more than one kind.* ***For Example****, The Address attribute is a composite attribute, multivalued attribute, stored attribute and a non-key attribute.*

RELATIONSHIPS AND RELATIONSHIP SETS

Relationship:

1. Relationship is an association between entities.
2. The entities that participate each relationship is identified by a name that describes the relationship



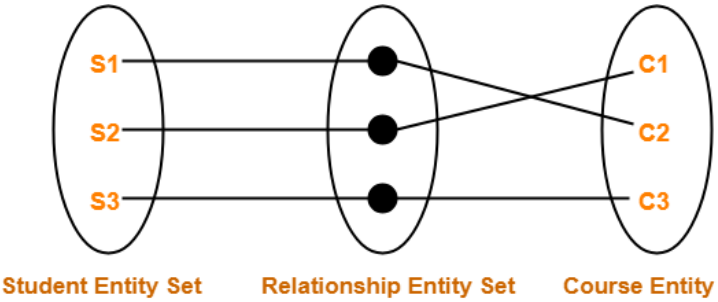
3. The relationship represents diamond symbol.
4. Relationships between entities always operate in both directions.

For example, A teacher teaches students. Here, "***teaches***" is a relationship and this is the relationship between a Teacher entity and a Student entity.

Relationship Set

A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

Example- Set representation of above ER diagram is-



Set Representation of ER Diagram

Degree of a Relationship Set- **The number of entity sets that participate in a relationship set is termed as the degree of that relationship set. Thus,**

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

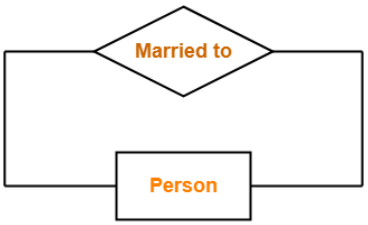
TYPES OF RELATIONSHIP SETS-

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

1. Unary relationship set
2. Binary relationship set
3. Ternary relationship set
4. N-ary relationship set

1. Unary Relationship Set- **Unary relationship set is a relationship set where only one entity set participates in a relationship set.**

Example- One person is married to only one person



Unary Relationship Set

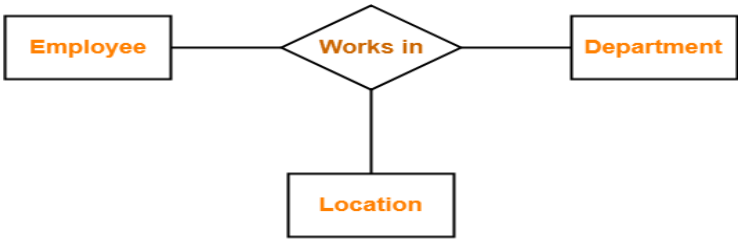
2. Binary Relationship Set- **Binary relationship set is a relationship set where two entity sets participate in a relationship set.**

Example- Student is enrolled in a Course

Binary Relationship Set

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

Example-



Ternary Relationship Set

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

MAPPING CARDINALITIES

Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

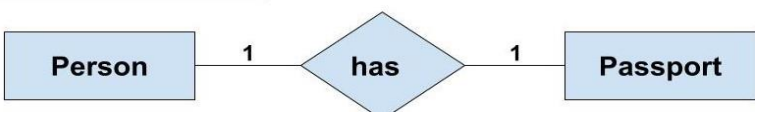
There are three types of relationships that can exist between two entities.

- One-to-One Relationship
- One-to-Many or Many-to-One Relationship
- Many-to-Many Relationship

One-to-One Relationship

Such a relationship exists when each record of one table is related to only one record of the other table.

For example, If there are two entities 'Person' (Id, Name, Age, Address)and 'Passport'(Passport_id, Passport_no). So, each person can have only one passport and each passport belongs to only one person.



Such a relationship is not very common. However, such a relationship is used for security purposes. In the above example, we can easily store the passport id in the 'Person' table only. But, we make another table for the 'Passport' because Passport number may be sensitive data and it should be hidden from certain users. So, by making a separate table we provide extra security that only certain database users can see it.

One-to-Many or Many-to-One Relationship

Such a relationship exists when each record of one table can be related to one or more than one record of the other table. This relationship is the most common relationship found. A one-to-many relationship can also be said as a many-to-one relationship depending upon the way we view it.

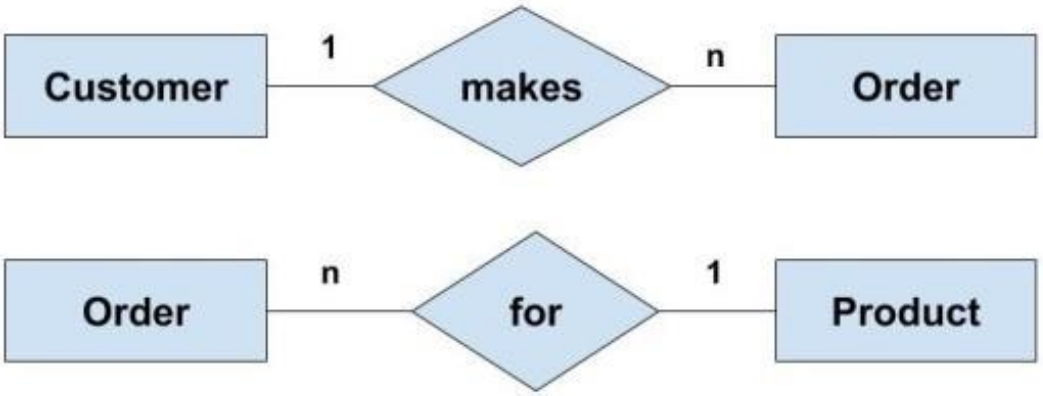
For example, If there are two entity type 'Customer' and 'Account' then each 'Customer' can have more than one 'Account' but each 'Account' is held by only one 'Customer'. In this example, we can say that each Customer is associated with many Account. So, it is a one-to-many relationship. But, if we see it the other way i.e many Account is associated with one Customer then we can say that it is a many-to-one relationship.

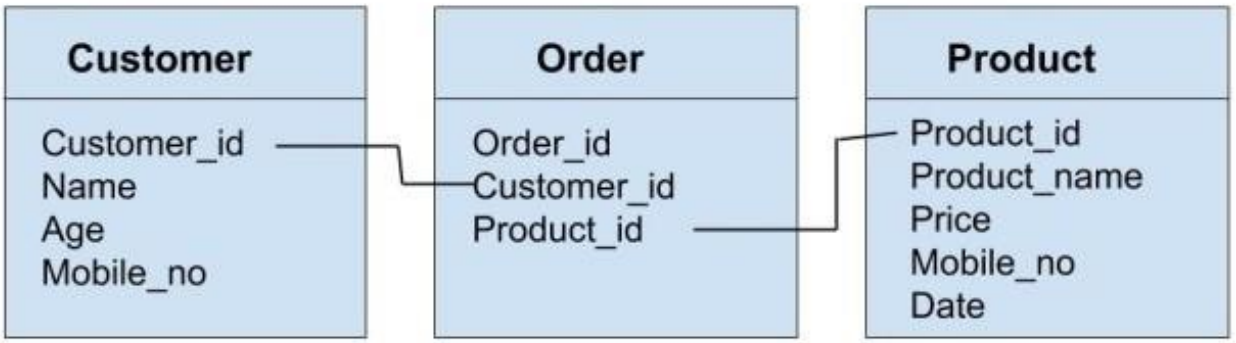
Many-to-Many Relationship

Such a relationship exists when each record of the first table can be related to one or more than one record of the second table and a single record of the second table can be related to one or more than one record of the first table. A many-to-many relationship can be seen as a two one-to-many relationship which is linked by a 'linking table' or 'associate table'. The linking table links two tables by having fields which are the primary key of the other two tables. We can understand this with the following example.

Example: If there are two entity type 'Customer' and 'Product' then each customer can buy more than one product and a product can be bought by many different customers.

Now, to understand the concept of the linking table here, we can have the 'Order' entity as a linking table which links the 'Customer' and 'Product' entity. We can break this many-to-many relationship in two one-to-many relationships. First, each 'Customer' can have many 'Order' whereas each 'Order' is related to only one 'Customer'. Second, each 'Order' is related only one Product wheres there can many orders for the same Product.





In the above concept of linking can be understood with the help of taking into consideration all the attributes of the entities 'Customer', 'Order' and 'Product'. We can see that the primary key of both 'Customer' and 'Product' entity are included in the linking table i.e 'Order' table. These key act as foreign keys while referring to the respective table from the 'Order' table.

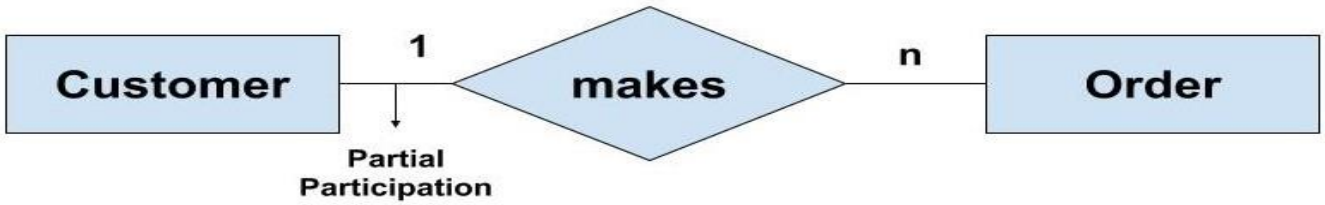
Participation Constraints

The relationship can be between two strong entity or a strong entity and a weak entity. Depending upon the type of entity participating in the relationship, the participation can be partial or total. There are two types of participation constraints:

- Partial Participation
- Total Participation

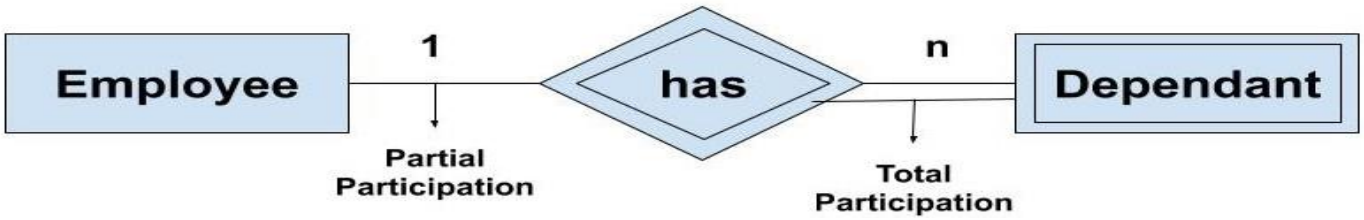
Partial Participation: Partial Participation exists when all the entity of an entity type is not associated with one or the other entity of another entity type. This is represented by joining the relationship with the entity type with the help of one line.

Example: We have two entity type 'Customer' and 'Order'. Then there can be 'Customer' who have not done any order. So, here there is partial participation of the entity in the relationship.



Total Participation: Total Participation exists when all the entity of an entity type is associated with one or the other entity of another entity type. This is represented by joining the relationship with the entity type with the help of a double parallel line. Such a relationship usually exist between a strong entity and a weak entity.

Example: We have two entity type 'Employee' and 'Dependant'. Then all the 'Dependent' entity are related to one or the other 'Employee' entity. This is called total participation of the entity in the relationship. But, it may be possible that some 'Employee' is not related to any of the 'Dependant' entity. So, 'Employee' is showing partial participation whereas the 'Dependant' is showing total participation in the relationship.



ADDITIONAL FEATURES OF ER:

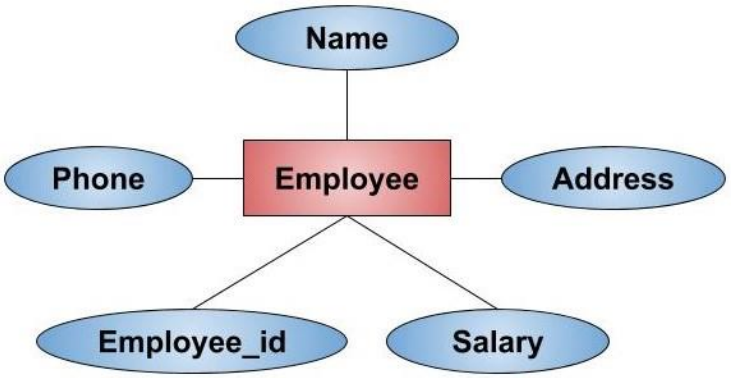
As the data is growing the complexity of managing these data is also increasing. We can't manage these data with the help of the traditional E-R data model. So to manage the increasing complexity we have designed an extended E-R Model. The extended E-R Model includes various concepts like Specialization, Generalization, Aggregation, etc.

Generalization

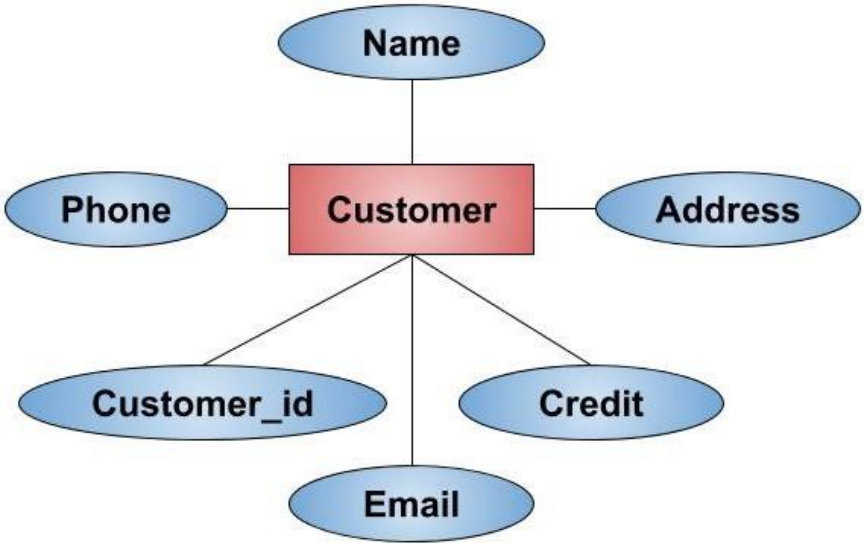
Generalization, as the name suggests, is a process of generalizing two or more lower-level entity types into a higher-level entity type. Entities are clubbed or grouped together to represent a more generalized view. In this process, the common attributes of two or more entities combine to form a new entity type. The new entity type formed is called a **generalized entity**. It may be possible that this generalized entity may combine further with other entity types to form another higher-level entity type. **It is a bottom-up approach**. It is the reverse process of **Specialization. For example,** Whales, Sharks, and Dolphins can be generalized as Fish. Similarly, Bicycle, Bike, and Car can be generalized as Vehicles.

Example: Suppose we have two entity types, Employee and Customer.

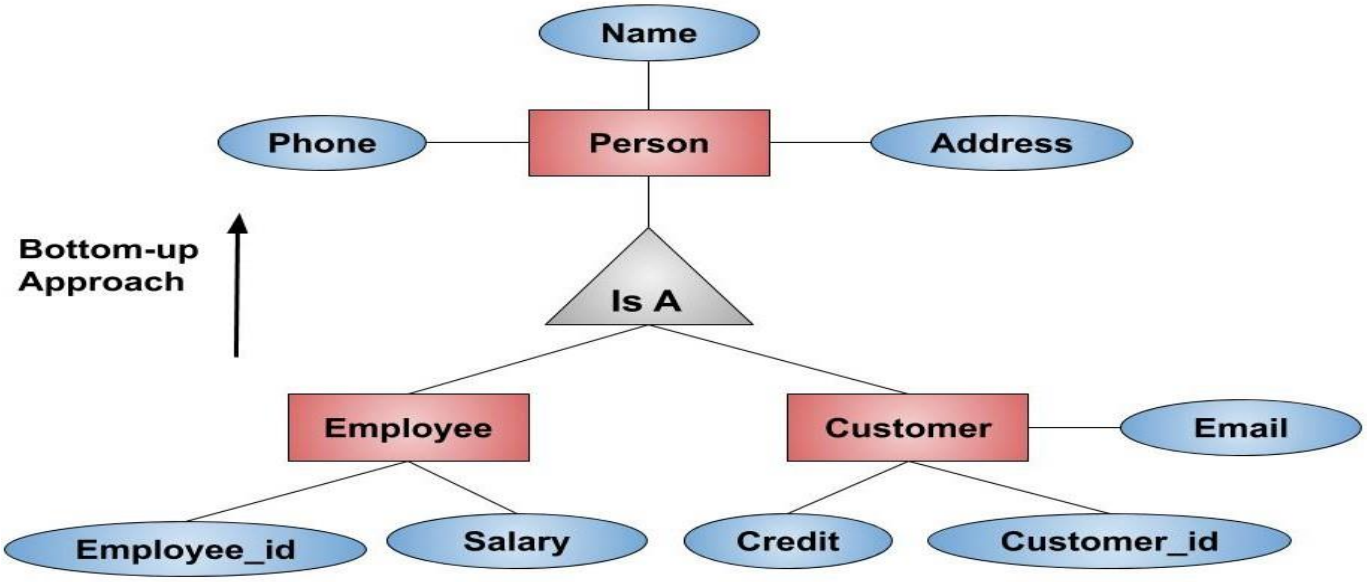
The attributes of Employee entity type are *Name, Phone, Salary, Employee_id,* and *Address*.



The attributes of Customer entity type are *Name, Phone, Address, Credit, Customer_id,* and *Email.*



We can see that the three attributes i.e. *Name, Phone,* and *Address* are common here. When we **generalize** these two entities, we form a new higher-level entity type Person. The new entity type formed is a **generalized entity.** We can see in the below E-R diagram that after generalization the new generalized entity Person contains only the common attributes i.e. *Name, Phone,* and *Address*. Employee entity contains only the specialized attribute like *Employee_id* and *Salary*. Similarly, the Customer entity type contains only specialized attributes like *Customer_id, Credit,* and *Email.* So from this example, it is also clear that when we go from bottom to top, it is a Generalization and when we go from top to bottom it is Specialization. Hence, we can say that Generalization is the reverse of Specialization.

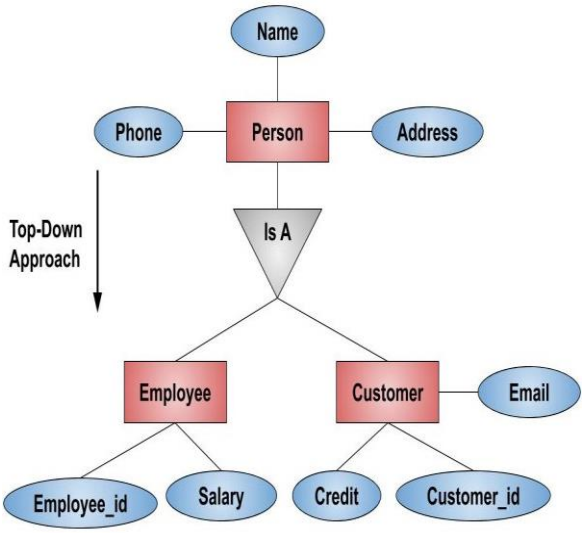


Generalization

Specialization

Specialization, as the name suggests, is a process of specializing an entity type into a more specified entity. In this process, we specialize a higher-level entity type by adding some additional attributes to the entity type. In this approach, we add some additional attributes and specialize a higher-level entity type into some other entity type. **It is a top-down approach in which a higher-level entity is broken into smaller entities**. ***For example,*** Animals can be mammals or reptiles. Then these mammals can either be tiger, elephant or humans. The reptiles can be snakes or lizards. So we are specializing as we are moving towards any particular type of animal.

Example: If we have a person entity type who has attributes such as *Name, Phone_no, Address*. Now, suppose we are making software for a shop then the person can be of two types. This Person entity can further be divided into two entity i.e. Employee and Customer. How we will specialize it? We will specialize the Person entity type to Employee entity type by adding attributes like *Emp_no and Salary*. Also, we can specialize the Person entity type to Customer entity type by adding attributes like *Customer_no, Email, and Credit.* These lower-level attributes will inherit all the properties of the higher-level attribute. The Customer entity type will also have attributes like *Name, Phone_no, and Address* which was present in the higher-level entity.

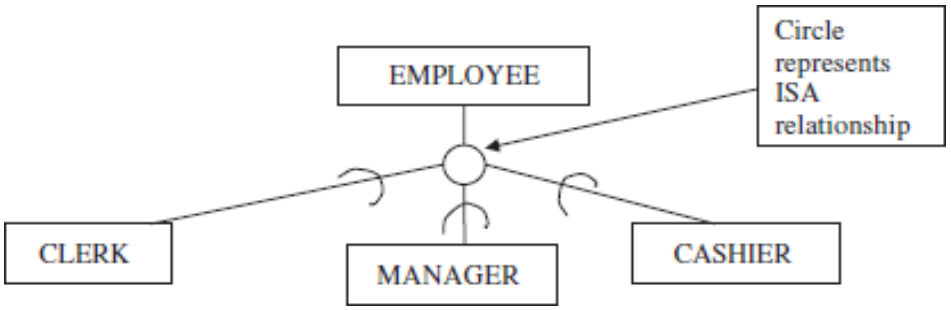


Specialization

ISA Relationship and Attribute Inheritance

IS A relationship supports attribute inheritance and relationship participation. In the EER diagram, the subclass relationship is represented by ISA relationship. Attribute inheritance is the property by which subclass entities inherit values for all attributes of the superclass.

Consider the example of EMPLOYEEentity set in a bank. TheEMPLOYEE in a bank can be CLERK, MANAGER, CASHIER, ACCOUNTANT, etc. It is to be observed that the CLERK, MANAGER, CASHIER, ACCOUNTANT inherit some of the attributes of the EMPLOYEE.



In this example the superclass is EMPLOYEE and the subclasses are CLERK, MANAGER, and CASHIER. The subclasses inherit the attributes of the superclass. Since each member of the subclass is an ISA member of the superclass, the circle below the EMPLOYEE entity set represents ISA relationship.

Multiple Inheritance *:* A subclass with more than one superclass is called a shared subclass. A subclass inherits attributes not only of its direct superclass, but also of all its predecessor superclass, that is it has multiple inheritance from its super classes. In multiple inheritance a subclass can be subclass of more than one superclass.

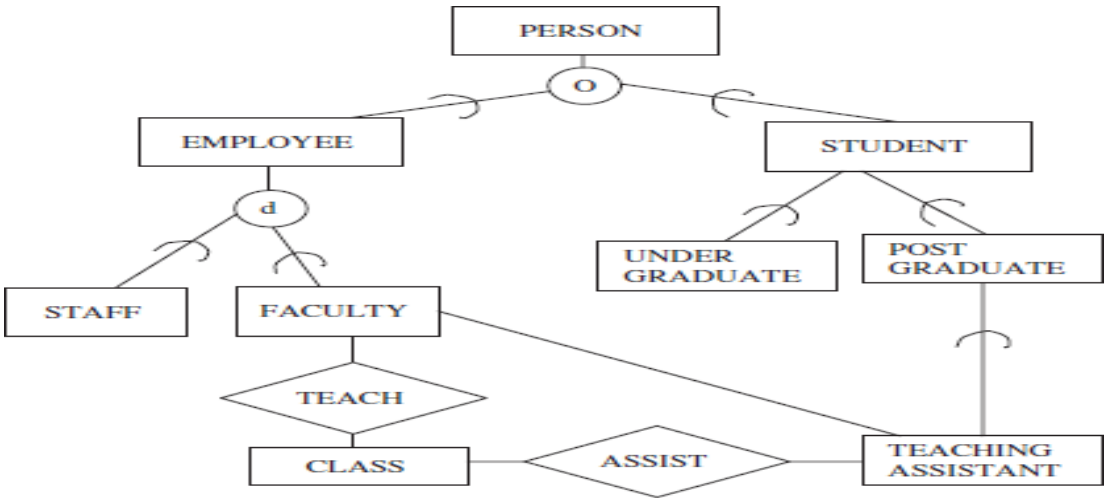


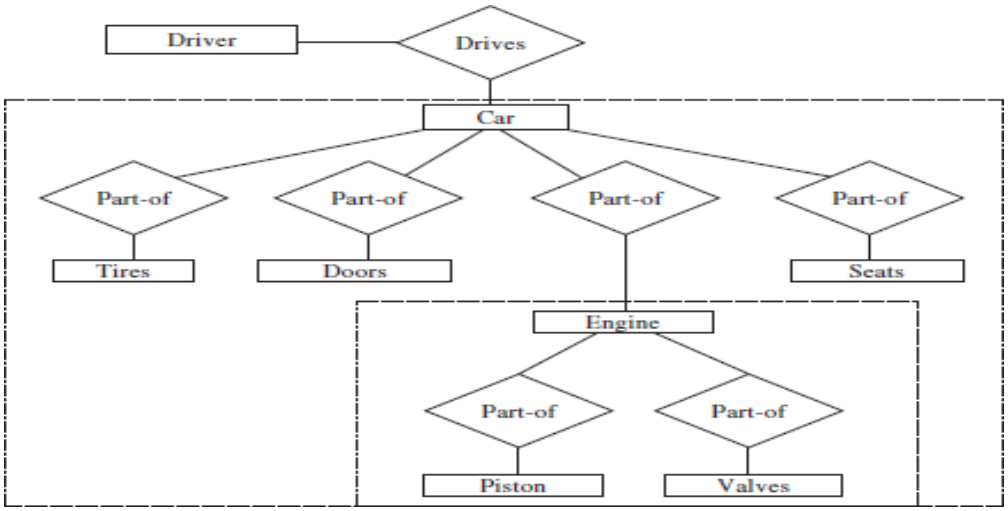
Fig. 2.2. Multiple inheritance

Aggregation and Composition

Relationships among relationships are not supported by the ER model. Groups of entities and relationships can be abstracted into higher level entities using aggregation. Aggregation represents a “HAS-A” or “IS-PART-OF” relationship between entity types. One entity type is the whole, the other is the part.

Aggregation allows us to indicate that a relationship set participates in another relationship set. Consider the example of a driver driving a car. The car has various components like tires, doors, engine, seat, etc., which varies from one car to another. Relationship drives is insufficient to model the complexity of this system. Part of relationships allow abstraction into higher level entities.

In this example engine, tires, doors, and seats are aggregated into car.



Composition is a stronger form of aggregation where the part cannot exist without its containing whole entity type and the part can only be part of one entity type.

Consider the example of DEPARTMENT has PROJECT. Each project is associated with a particular DEPARTMENT. There cannot be a PROJECT without DEPARTMENT. Hence DEPARTMENT has PROJECT is an example of composition.

CONCEPTUAL DESIGN WITH ER MODEL

Conceptual design is the first stage in the database design process. The goal at this stage is to design a database that is independent of database software and physical details. The output of this process is a conceptual data model that describes the main data entities, attributes, relationships, and constraints of a given problem domain. This design is descriptive and narrative in form. Keep in mind the following minimal data rule:

"All that is needed is there, and all that is there is needed".

In other words, make sure that all data needed are in the model and that all data in the model are needed. All data elements required by the database transactions must be defined in the model, and all data elements defined in the model must be used by at least one database transaction. The conceptual design has four steps, which are as follows.

1. Data analysis and requirements
2. Entity relationship modeling and normalization
3. Data model verification
4. Distributed database design

1. **Data Analysis and Requirements**:

The first step in conceptual design is to discover the characteristics of the data elements. Appropriate data element characteristics are those that can be transformed into appropriate information. Therefore, the designer's efforts are focused on:

- Information needs. What kind of information is needed—that is, what output (reports and queries) must be generated by the system, what information does the current system generate, and to what extent is that information adequate?
- Information users. Who will use the information? How is the information to be used? What are the various end-user data views?

• Information sources. Where is the information to be found? How is the information to be extracted once it is found?
• Information constitution. What data elements are needed to produce the information? What are the data attributes? What relationships exist among the data? What is the data volume? How frequently are the data used? What data transformations are to be used to generate the required information? The designer obtains the answers to those questions from a variety of sources in order to compile the necessary information.

Note these sources:
• Developing and gathering end-user data views. The database designer and the end user(s) interact to jointly develop a precise description of end-user data views. In turn, the end-user data views will be used to help identify the database's main data elements.

• Directly observing the current system: existing and desired output. The end user usually has an existing system in place, whether it's manual or computer-based. The designer reviews the existing system to identify the data and their characteristics.

• Interfacing with the systems design group. The database design process is part of the Systems Development Life Cycle (SDLC). In some cases, the systems analyst in charge of designing the new system will also develop the conceptual database model.

2. Entity Relationship Modeling and Normalization:

Before creating the ER model, the designer must communicate and enforce appropriate standards to be used in the documentation of the design. The process of defining business rules and developing the conceptual model using ER diagrams can be described using the following steps.

1. Identify, analyze, and refine the business rules.
2. Identify the main entities, using the results of Step 1.
3. Define the relationships among the entities, using the results of Steps 1 and 2.
4. Define the attributes, primary keys, and foreign keys for each of the entities.
5. Normalize the entities. (Remember that entities are implemented as tables in an RDBMS.)
6. Complete the initial ER diagram.
7. Validate the ER model against the end users' information and processing requirements.
8. Modify the ER model, using the results of Step 7.

3. Data Model Verification:

The data model verification step is one of the last steps in the conceptual design stage, and it is also one of the most critical ones. In this step, the ER model must be verified against the proposed system processes in order to corroborate that the intended processes can be supported by the database model. Verification requires that the model be run through a series of tests against:

• End-user data views.
• All required transactions: SELECT, INSERT, UPDATE, and DELETE operations.
• Access rights and security.
• Business-imposed data requirements and constraints.

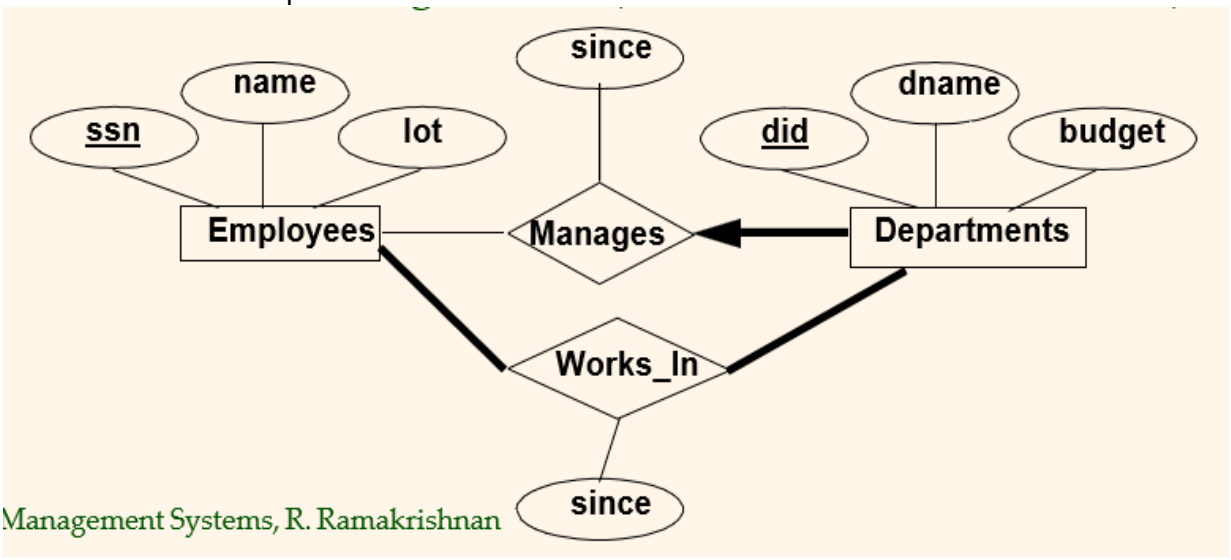
4. Distributed Database Design:

Although not a requirement for most databases, sometimes a database may need to be distributed among multiple geographically disperse locations. Processes that access the database may also vary from one location to another. For example, a retail process and a warehouse storage process are likely to be found in different physical locations. If the database data and processes are to be distributed across the system, portions of a database, known as database fragments, may reside in several physical locations.

Conceptual design with ER: Steps in Designing an Entity-Relationship Schema

- Identify entity types (entity type vs. attribute)
- Identify relationship types
- Identify and associate attributes with entity and relationship types
- Determine attribute domains
- Determine primary key attributes for entity types
- Associate (refined) cardinality ratio(s) with relationship types
- Design generalization/specialization hierarchies including constraints (includes natural language statements as well)

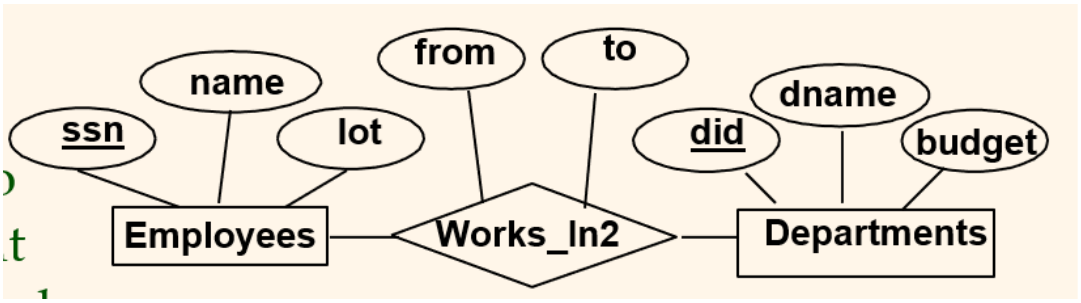
Consider the below example



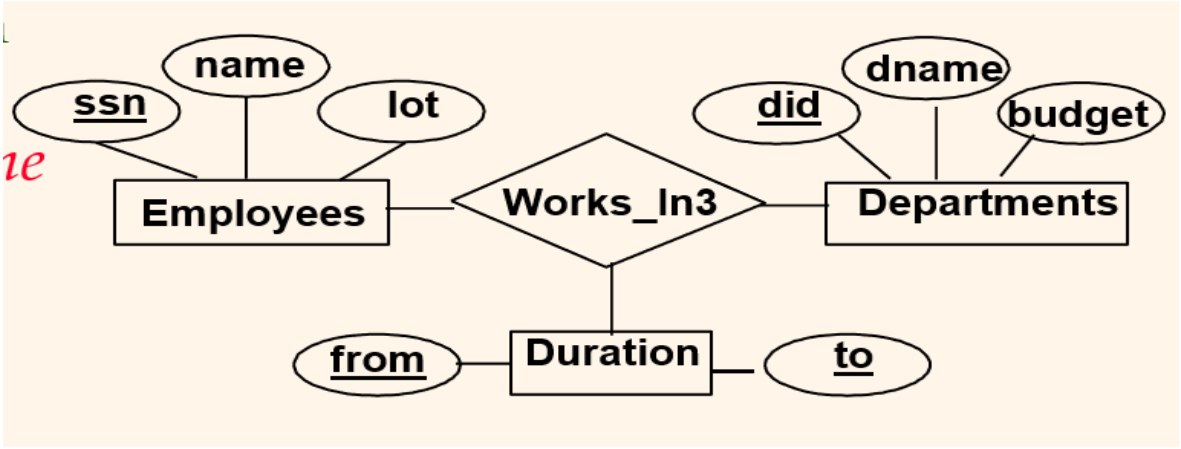
1. Choosing Entity Set vs Attributes

how choosing an entity set vs an attribute can change the whole ER design semantics. To understand this lets take an example,

- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
- Depends upon the use we want to make of address information, and the semantics of the data:
  - If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).
  - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modelled as an entity (since attribute values are atomic).
- Works_In2 does not allow an employee to work in a department for two or more periods



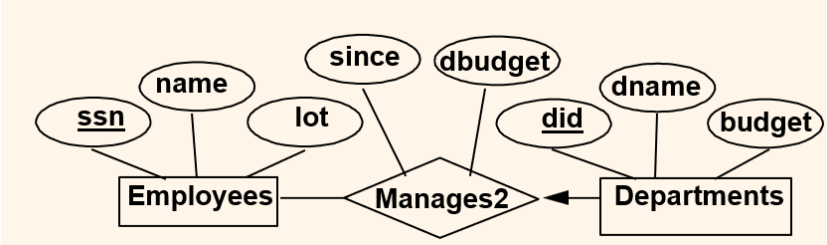
- Similar to the problem of wanting to record several addresses for an employee: we want to record *several values of the descriptive attributes for each instance of this relationship.*



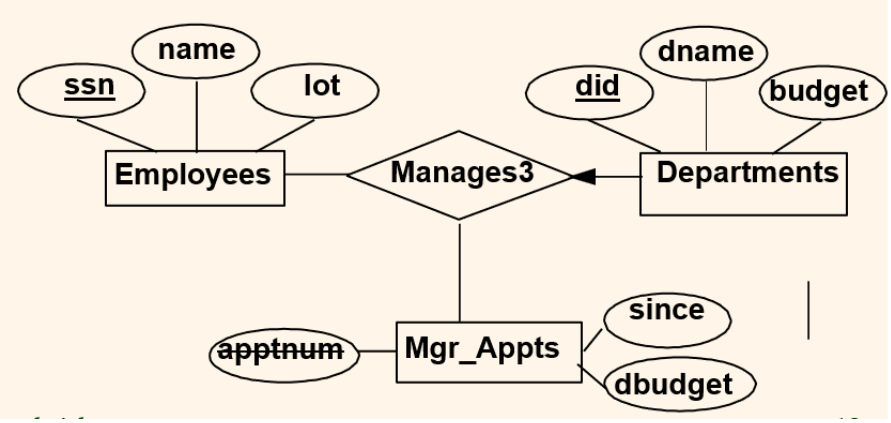
2. Choosing Entity Set vs. Relationship Sets

It is hard to decide that an object can be best represented by an entity set or relationship set. To comprehend and decide the perfect choice between these two (entity vs relationship), the user needs to understand whether the entity would need a new relationship if a requirement arise in future, if this is the case then it is better to choose entity set rather than relationship set.

- First ER diagram OK if a manager gets a separate discretionary budget for each dept.



- What if a manager gets a discretionary budget that covers *all* managed depts?
  Redundancy of *dbudget,* which is stored for each dept managed by the manager
  Misleading: suggests *dbudget* tied to managed dept



3. Choosing Binary vs n-ary Relationship Sets

In most cases, the relationships described in an **ER diagrams** are binary. The **n-ary** relationships are those where entity sets are more than two, if the entity sets are only two, their relationship can be termed as binary relationship.

The n-ary relationships can make ER design complex, however the good news is that we can convert and represent any n-ary relationship using multiple binary relationships.

This may sound confusing so lets take an example to understand how we can convert an n-ary relationship to multiple binary relationships. Now lets say we have to describe a relationship between four family members: father, mother, son and daughter. This can easily be represented in forms of multiple binary relationships, father-mother relationship as “spouse”, son and daughter relationship as “siblings” and father and mother relationship with their child as “child”.

4. Placing Relationship Attributes

The **cardinality ratio** in DBMS can help us determine in which scenarios we need to place relationship attributes. It is recommended to represent the attributes of **one to one** or **one to many** relationship sets with any participating entity sets rather than a relationship set.

For example, if an entity cannot be determined as a separate entity rather it is represented by the combination of participating entity sets. In such case it is better to associate these entities to many-to-many relationship sets.

5. Functional dependencies:

1. *e.g., A dept can’t order two distinct parts from the same supplier*.
    1. Can’t express this wrt ternary Contracts relationship.

Normalization refines ER design by considering FDs.

Inclusion dependencies:

– Special case: Foreign keys (ER model can express these).

– *e.g., At least 1 person must report to each manager.* (Set of *ssn* values in Manages must be subset of *supervisor_ssn* values in Reports_To.) Foreign key? Expressible in ER model?

- General constraints:

– *e.g., Manager’s discretionary budget less than 10% of the combined budget of all departments he or she manages*.

Module 2: RELATIONAL MODEL AND RELATIONAL ALGEBRA (08 Periods)

Relational Model: Creating and modifying relations, Integrity constraints over relations, Enforcing integrity constraints, Querying relational data, Logical database design, Introduction to views, Destroying/altering tables and views.

Relational Algebra: Preliminaries, Relational Algebra operators.

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 Data Model:

1. The relational model uses a collection of tables to represent both data and relationships.
2. Tables are logical structures maintained by the database manager.
3. The relational model is a combination of three components,
    a. Structural
    b. Integrity and
    c. Manipulative parts.

a. Structural Part : *The structural part defines the database as a collection of relations.*

b. Integrity Part : *The database integrity is maintained in the relational model using primary and foreign keys.*

c. Manipulative Part

1. The relational algebra and relational calculus are the tools
2. It used to manipulate data in the database.
3. Thus relational model has a strong mathematical background.

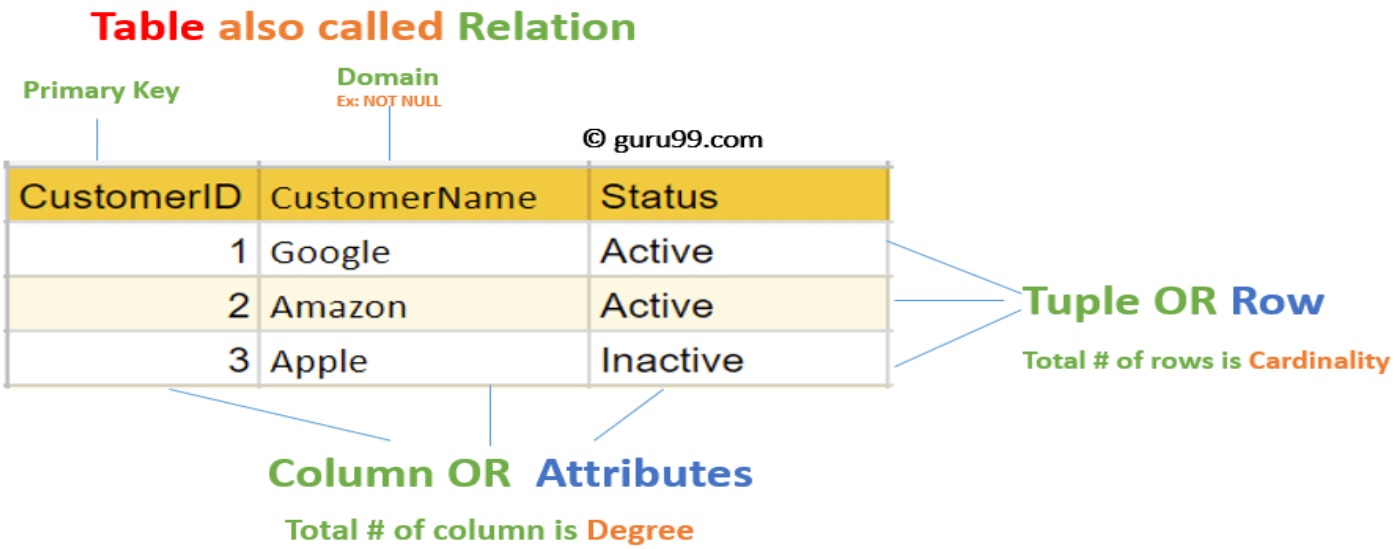
Some popular Relational Database management systems are:

- DB2 and Informix Dynamic Server – IBM
- Oracle and RDB – Oracle
- SQL Server and Access – Microsoft

Relational Model Concepts: This database consists of various components based on the relational model. These include:

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.



CREATING AND MODIFYING RELATIONS USING SQL

A database query refers to the request of data or information from a database. A database query is performed using a database query language that generates data of different formats according to function. A number of query languages have been developed according to different database engines and purposes, one such language is SQL. SQL stands for Structured Query Language. It is the most widely used and well-known database query language which generates result in the form of rows and columns. It is designed for the retrieval and management of data in a relational database.

SQL can be used to perform the following operations in the database :

- Access data in relational database management systems.
- Describe the data.
- Manipulate that data.
- Create and drop databases and tables.
- Create a view, stored procedure, functions in a database.
- Set permissions on tables, procedures and views

Create and modifying the relationships:

Create: Create is used to create schema, tables, index, and constraints in the database. The basic syntax to create table is as follows.

Syntax: CREATE TABLE table_name (column1 data_type(size), column2 data_type(size), ..columnN data_type(size));

Ex1 : CREATE TABLE Employees (Emp_Id int(3), Emp_Name varchar(20),address varchar(20),dept_id int);

Insert : Insert statement is used to insert new records into the table.

Syntax1: INSERT INTO TABLE_NAME (col1, col2, col3,...colN) VALUES (value1, value2, value3,...valueN);

It inserts value to each column from 1 to N. When we insert the data in this way, we need to make sure datatypes of each column matches with the value we are inserting. Else, data will not be inserted or will insert wrong values. Also, if there is any foreign key constraint on the table, then we have to make sure foreign key value already exists in the parent table.

Ex: INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, ADDRESS, DEPT_ID) VALUES
(10001, 'Joseph', 'Troy', 11101); -
Here dept_id 11101 is a foreign key and it has be inserted into department table before inserting into the employee table.

Syntax2: INSERT INTO table_name VALUES(value1, value2,… valueN);

We can insert the values into the table without specifying the columns, provided we enter the value in the order the table structure is. Imagine table structure for Employee is as follows and then we can insert the record without specifying column names as below:

Ex: INSERT INTO EMPLOYEE VALUES (10001, 'Joseph', 'Troy', 11101);

If we are specifying the column list then we need not insert it in the order of table structure. It inserts the values in the order the column is listed in the INSERT statement.

INSERT INTO EMPLOYEE (EMP_ID, DEPT_ID, ADDRESS, EMP_NAME) VALUES (10001, 11101,'Troy', 'Joseph');

We can even copy some of the column values from the existing table. Suppose we have to copy emp_id, emp_name from Employee table to some temporary table. Then we can write as follows: (Assuming here that datatype in both the tables are same and emp_name has enough buffers to store both first name and last name combined).

```
INSERT INTO TEMP_EMP
SELECT e.EMP_ID, e.EMP_FIRST_NAME || ' '|| e.EMP_LAST_NAME as emp_name
FROM EMPLOYEE e;
```

Update : The update command is used to update or modify the existing data of a table in the database. It changes the values of the records. Using UPDATE command we can update a single column as well as multiple columns simultaneously.

Syntax: UPDATE table_name SET col1 = value1, col2 = value2,… colN = valueN WHERE condition;

table_name: name of the table whose data we are going to update.
SET: keyword used to specify fields and values.
col: name of fields to be updated.
value: the updated value of the corresponding field.
condition: condition specified to select rows in which data needs to be modified.

If we do not specify 'WHERE' condition in the 'UPDATE' statement, then it will update the whole table. Hence we have to specify which record/employee has to be updated.

Ex: UPDATE Employees SET Salary = 10000 WHERE Emp_Id = 007;
Modifies the salary(single column) of the employee with Emp_Id 7 and sets Salary to 10000.

UPDATE Employees SET Salary=10000, Name="John" WHERE Emp_Id = 007;
Modifies the salary and name (multiple-column) of employees with Emp_Id 7 and sets Salary to 10000.

Delete : Using Delete statement, we can delete the records in the entire table or specific record by specifying the condition.

Syntax: DELETE FROM table_name [WHERE condition];
If we do not specify the condition, then it would delete entire record from the table

Ex: Suppose we have to delete an employee with id 110 from Employee table. Then the delete statement would be
DELETE FROM EMPLOYEE WHERE EMP_ID = 110;

INTEGRITY CONSTRAINTS OVER RELATIONS

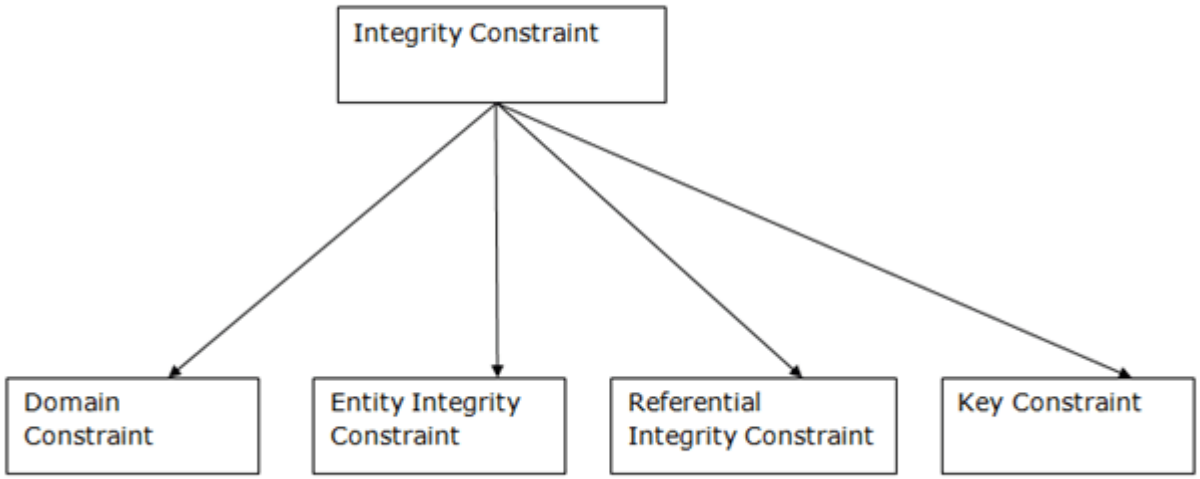
Data Integrity is having correct and accurate data in your database. When we are storing data in the database we don't want repeating values, we don't want incorrect values or broken relationships between tables.

For any stored data if we want to preserve the consistency and correctness, a relational DBMS typically imposes one or more data integrity constraints. These constraints restrict the data values which can be inserted into the database or created by a database update.

Data Integrity Constraints

There are different types of data integrity constraints that are commonly found in relational databases, including the following −

- **Required data** − Some columns in a database contain a valid data value in each row; they are not allowed to contain NULL values. In the sample database, every order has an associated customer who placed the order. The DBMS can be asked to prevent NULL values in this column.
- **Validity checking** − Every column in a database has a domain, a set of data values which are legal for that column. The DBMS allowed preventing other data values in these columns.
- **Entity integrity** − The primary key of a table contains a unique value in each row that is different from the values in all other rows. Duplicate values are illegal because they are not allowing the database to differentiate one entity from another. The DBMS can be asked to enforce this unique values constraint.
- **Referential integrity** − A foreign key in a relational database links each row in the child table containing the foreign key to the row of the parent table containing the matching primary key value. The DBMS can be asked to enforce this foreign key/primary key constraint.
- **Other data relationships** − The real-world situation which is modeled by a database often has additional constraints which govern the legal data values that may appear in the database. The DBMS is allowed to check modifications to the tables to make sure that their values are constrained in this way.
- **Business rules** − Updates to a database that are constrained by business rules governing the real-world transactions which are represented by the updates.
- **Consistency** − Many real-world transactions that cause multiple updates to a database. The DBMS is allowed to enforce this type of consistency rule or to support applications that implement such rules.



Domain Integrity :

Domain refers to the range of acceptable values. It refers to the range of values that we are going to accept and store in a particular column within a database. The data types available are mainly integer, text, date etc. Any entry which we make for a column should be available in the domain of the data type.

Example: If we have to store the salary of the employees in the '*employee_table*' then we can put constraints that it should only be an INTEGER. Any entry other than integer like characters would not be acceptable and when we try to give input like this, the DBMS will produce errors.

Employee_id	Name	Salary	Age
1	Andrew	486522	25
2	Angel	978978	30
3	Anamika	697abc	35

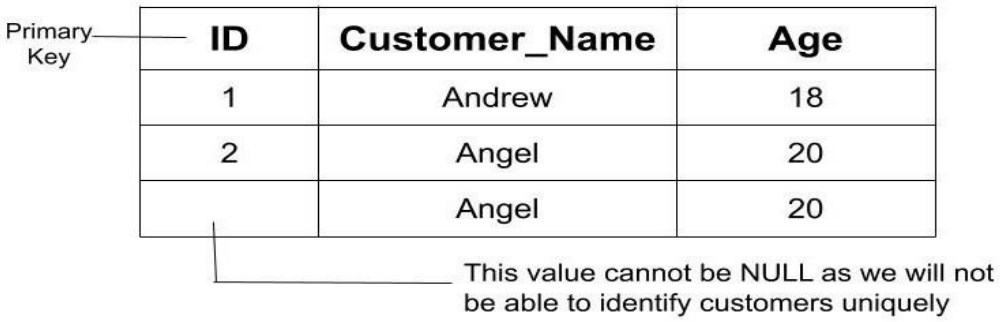
This value is out of domain(not INTEGER)so it is not acceptable.

DOMAIN INTEGRITY

Entity Integrity

Each row for an entity in a table should be uniquely identified i.e. id some record is saved in the database then that record should be uniquely identified from others. This is done with the help of primary keys. The entity constraint says that the value of the primary key should not be NULL. If the value of the primary key is NULL then we can't uniquely identify the rows if all other fields are the same. Also, with the help of primary key, we can uniquely identify each record.

Example: 1 If we have a customer database and customer_table is present there with attributes like age and name. Then each customer should be uniquely identified. There might be two customers with the same name and same age, so there might be confusion while retrieving the data. If we retrieve the data of customer named 'A*ngel*' then two rows are having this name and there would be confusion. So, to resolve this issues primary keys are assigned in each table and it uniquely identifies each entry of the table.



ENTITY INTEGRITY

Example: 2

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

Referential Integrity Constraints

It states that if a foreign key exists in a relation then either the foreign key value must match a primary key value of some tuple in its home relation or the foreign key value must be null.

The rules are:

1. You can't delete a record from a primary table if matching records exist in a related table.
2. You can't change a primary key value in the primary table if that record has related records.
3. You can't enter a value in the foreign key field of the related table that doesn't exist in the primary key of the primary table.
4. However, you can enter a Null value in the foreign key, specifying that the records are unrelated.

EXAMPLE-

Consider 2 relations "stu" and "stu_1" Where "Stu_id " is the primary key in the "stu" relation and foreign key in the "stu_1" relation.

Relation "stu"

Stu_id	Name	Branch
11255234	Aman	CSE
11255369	Kapil	EcE

11255324	Ajay	ME
11255237	Raman	CSE
11255678	Aastha	ECE

Relation "stu_1"

Stu_id	**Course**	**Duration**
11255234	B TECH	4 years
11255369	B TECH	4 years
11255324	B TECH	4 years
11255237	B TECH	4 years
11255678	B TECH	4 years

Examples

Rule 1. You can't delete any of the rows in the ”stu” relation that are visible since all the ”stu” are in use in the “stu_1” relation.

Rule 2. You can't change any of the ”Stu_id” in the “stu” relation since all the “Stu_id” are in use in the ”stu_1” relation. * Rule 3.* The values that you can enter in the” Stu_id” field in the “stu_1” relation must be in the” Stu_id” field in the “stu” relation.

Rule 4 You can enter a null value in the "stu_1" relation if the records are unrelated.

Example2 : Let us suppose we have two tables of the student(student_id, name, age, course_id) and course(course_id, course_name, duration). Now, if any course_id is present in the student table which is not there in the course table then this is not allowed. The course_id in the student table should either be null or if any course_id is present in the student table then it should also be present in the course table. This is how referential integrity is maintained.

Student (First Table)

Roll_no	Student_name	Age	Course_id
1	Andrew	18	78
2	Angel	19	16
3	Priya	20	56
4	Analisa	21	

Foreign Key

This value is not allowed because this value is not defined as a primary key in the course table.

The value can be NULL as the student(Analisa) may not have taken any course.

Primary Key

Course (Second Table)

Primary Key

Course_id	Course_name	Duration (months)
78	Big Data	4
56	Algorithm	2

REFERENTIAL INTEGRITY

ENFORCING INTEGRITY CONSTRAINTS

SQL Constraints are the rules or restrictions applied to the database to limit the type and accuracy of data entering the table. If the data to enter satisfies the constraints rule, it enters successfully. **Let's consider** the table named **Employee** with EmpId as the primary key.

EmpId	Last_Name	First_Name	Title	Salary	Location
1	Bharti	Nisha	Technical Staff	80000	Bangalore
2	Jha	Durgesh	Data Analyst	55000	Hyderabad
3	Sharma	Esha	Consultant	70000	Mumbai
4	Ahmed	Sarfaraz	Associate Consultant	65000	Chennai
5	Bharti	Amish	Software Developer	82000	Gurgaon

Let's consider another table named **Designation**. It contains the list of all the available designations in the company.

D_Id	Column 2
1	Associate Consultant
2	Consultant
3	Data Analyst
4	Software Developer
5	Technical Staff

We will insert a new value in the table and check if they violate the constraints.

`INSERT into Employee VALUES (6, 'Patra', 'Suman', 'Business Analyst', 90000, 'Pune');`

Here, you can see that the designation Business Analyst is not available in the company list. So, none of the employees can have a designation other than that present in the list. Entering any **value other than what satisfies** the constraints rule would lead to a violation.

EmpId	Last_Name	First_Name	Title	Salary	Location
1	Bharti	Nisha	Technical Staff	80000	Bangalore
2	Jha	Durgesh	Data Analyst	55000	Hyderabad
3	Sharma	Esha	Consultant	70000	Mumbai
4	Ahmed	Sarfaraz	Associate Consultant	65000	Chennai
5	Bharti	Amish	Software Developer	82000	Gurgaon
6	Patra	Suman	Business Analyst	90000	Pune

Integrity constraint comes into action when the primary key gets references by a foreign key. It defines that the values of the foreign key must be present in the **primary key or is NULL**. There are three primary causes of referential integrity constraint violation. They are as follows:

- **Insertion** of new tuples in a referencing relation in a database
- **Deletion** of some existing tuples from a referenced relation in the database
- **Updation** or changes in the values of the existing tuples in a referenced relation in the database

Applying the above modifications to the relations in the database may or may not violate the constraints of the database.

Insertion in a Referencing Relation :

Insertion of a tuple in the relationship can cause violations in different ways. Let's have a look **at each of them one by one.**

Violation of Domain Constraint : Domain constraint in SQL restricts the values or type of attributes a column can contain. The data type of values of a domain can be string, **integer, currency, character, date, or time data type**.

Let's see examples of domain constraints.

- For representing the age in a database, the value inserted should always be a number. Insertion of any character or string or another data type may cause a violation of domain constraint.
- The domain constraint not only restricts the datatype but also defines the range or set of values of the attribute. For example, the domain constraint can fix the age between 10 to 50. Another number inserted greater than 50 or less than10 causes domain constraint violation.

Violation of the Entity integrity constraint : The entity constraint makes sure that none of the values in the primary key column is **assigned NULL**. The **primary key identifies** the individual records and connects the relations and thus cannot be NULL.

Violation of Key Constraints : **Keys in DBMS** are attributes or sets of attributes that uniquely identify rows in the relation. The primary key is one of the multiple keys present in the entity set that can contain unique and non NULL values in the database. Insertion of value violates the key constraints when it is already present in an **existing tuple in the table**.

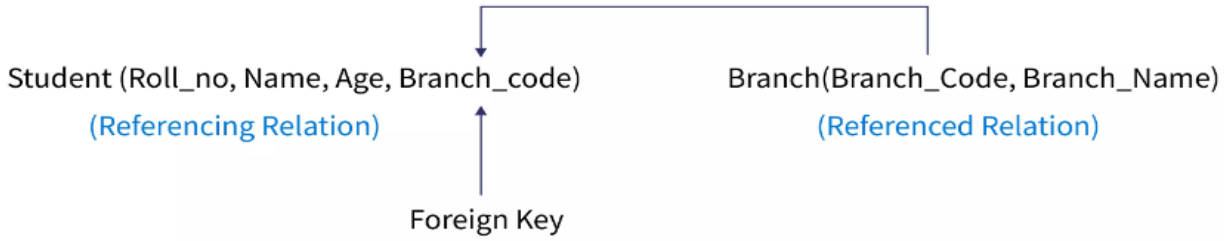
Violation of Referential Integrity Constraints : Values already present in the list of referenced values are **allowed to insert** in the referencing attribute.

Values not present in the list of referenced values cannot get inserted in the referencing attribute. **Inserting such values causes referential integrity constraint violation**.

Example: We will discuss with the help of the following two schemas:

Student (Roll_no, Name, Age, Branch_Code)

Branch(Branch_Code, Branch_Name)



The Student schema is the referencing relation, while the Branch schema is the referenced relation. It can also be said as the Student relation references the Branch relation. Branch_Code is the foreign key in the tables.

Student

Roll_no	Name	Age	Branch
1	Nisha	22	CSE
2	Suman	23	CE
3	Durgesh	21	MME
4	Sarfaraz	24	CSE
5	Yashwant	23	EE

Branch

Branch_Code	Branch_Name
CSE	Computer Science
ECE	Electronics Engineering
EE	Electrical Engineering
IT	Information Technology
CE	Civil Engineering
MME	Metallurgy and Materials Engineering

According to the constraints, any value not present in the referenced table cannot get **inserted into the referencing table**. For the above table, Students having Branch_Code other than the values in the Branch table cannot get inserted into the Students table.

For example, a Student with Branch_Code BT (BioTechnology) cannot be present in the Students relation as the **branch code BT** is not present in the relation Branch.

Deletion from a Referenced Relation

Deletion of values from a table can only cause **referential integrity constraint violation**.

According to the constraints, if the referencing attribute uses a value of the referenced attribute, that row cannot get deleted from the referenced relation. Deleting such rows can cause referential integrity constraint violation.

Let's consider some examples from the above two relations that cause integrity constraint violation.

- A row having Branch_Code CSE (Computer Science and Engineering) cannot delete from the relation Branch. It is because referencing relation Student references the value 'CSE' by the referencing attribute Branch_Code.

`DELETE FROM Branch WHERE Branch_Code = 'CSE';`

The above SQL statement deletes the record of the computer science branch from the Branch table. It violates the constraints as there is already a student present in the Student table with its Branch as CSE.

Student

Roll_no	Name	Age	Branch
1	Nisha	22	**CSE**
2	Suman	23	CE
3	Durgesh	21	MME
4	Sarfaraz	24	CSE
5	Yashwant	23	EE

Branch

Branch_Code	Branch_Name
ECE	Electronics Engineering
EE	Electrical Engineering
IT	Information Technology
CE	Civil Engineering
MME	Metallurgy and Materials Engineering

- A row having Branch_Code ECE (Electronics Engineering) can be safely deleted from the relation Branch as the referencing relation Student does not reference the value 'ECE' by the referencing attribute Branch_Code.

`DELETE FROM Branch WHERE Branch_Code = 'ECE';`

The above SQL statement deletes the record of the Electronics Engineering branch from the Branch table. Deleting that doesn't violate the constraints as there is no record in the Student table referenced by it.

Student

Roll_no	Name	Age	Branch
1	Nisha	22	CSE
2	Suman	23	CE
3	Durgesh	21	MME
4	Sarfaraz	24	CSE
5	Yashwant	23	EE

Branch

Branch_Code	Branch_Name
EE	Electrical Engineering
IT	Information Technology
CE	Civil Engineering
MME	Metallurgy and Materials Engineering

Handling the Violations : The referential integrity constraint violation caused by the deletion from a referenced relation gets handled in the **following three ways:**

- It is by deleting the tuples from the referencing relation simultaneously where the referencing attribute has the value of the referenced attribute to be deleted. This way of constraint violation **handling is known as On Delete Cascade**.

- The second method is to abort or delete the deletion request of the referenced relation if the value to delete is present in the **referencing attribute of the referencing relation**.
- The third method involves replacing the value with **NULL or some other value** in the referencing relation if the referencing attribute uses the value to delete from the referenced relation.

Updation in a Referenced Relation

According to the constraints, a row of the referenced relation cannot update if the referencing attribute uses the value of the referenced attribute of the row. Updating such rows going against the constraints causes the referential integrity constraint violation.

Let's see an example considering the above two relations.
`UPDATE Branch SET Branch_Code = 'CS' WHERE Branch_Code = 'CSE';`

The Branch_Code of a tuple in the relation Branch cannot update from CSE to CS as the referencing attribute Branch_Code references value CSE in the referencing relation Student.

Handling the Violations

The referential integrity constraint violation caused by the **update from a referenced relation** gets handled in the following three ways:

- It is by updating the tuples from the referencing relation simultaneously where the referencing attribute has the value of the referenced attribute to be updated. This way of constraint violation handling is **known as On Update Cascade.**
- The second method is to abort or delete the deletion request of the referenced relation if the value to update is present in the referencing attribute of the **referencing relation.**
- The third method involves replacing the value with NULL or other value in the referencing relation if the referencing attribute uses the value updated in the **referenced relation**.

QUERYING RELATIONAL DATA

Select

Select command helps to pull the records from the tables or views in the database. It either pulls the entire data from the table/view or pulls specific records based on the condition. It can even retrieve the data from one or more tables/view in the database.

Syntax 1: SELECT * FROM table_name; -- retrieves all the rows and columns from table table_name and displays it in tabular form.

Ex: SELECT * FROM STUDENT; -- All the columns are retrieved and displayed in below format

Syntax 2: SELECT COLUMN1, COLUMN2, COLUMN3 from table_name; -- retrieves only 3 columns from table table_name

Ex : SELECT STUDENT_NAME, ADDRESS FROM STUDENT; -- Retrieves only name and address from STUDENT table

Syntax 3: SELECT t1.COLUMN1, t2.COLUMN1 FROM table_name1 t1, table_name2 t2
WHERE t1.COLUMN2 = t2.COLUMN2; -- Combines 2 tables and retrieves specific columns from both the tables.

Ex: SELECT e.EMPLOYEE_NAME, e.ADDRESS, e.PHONE_NUMBER, d.DEPARTMENT_NAME FROM EMPLOYEE e,

DEPARTMENT d WHERE E.DEPARTMENET_ID = D.DEPARTMENT_ID; -- Displays employee name, address, phone number and the department name for which he works, by joining EMPLOYEE and DEPARTMENT tables.

General syntax of SELECT is:

SELECT column_list FROM table-name [WHERE Clause] [GROUP BY clause] [HAVING clause] [ORDER BY clause];

WHERE Clause – here we can specify the filter conditions to the query. We can add any number of conditions.

The WHERE clause is used to filter certain records that meet the conditions mentioned in the query. It is evaluated second after the FROM clause.

We can use the following operators with WHERE clause.

Operator	Description
=	Equal
>=	Greater than equal to
<=	less than equal to
>	Greater than
<	Less than
<>	Not equal
BETWEEN	Between a range
Like	Search for a pattern

GROUP BY – We can combine specific categories of columns together and show the results. For example, there are multiple employees working in different department. Using group by option we can display how many employees are working in each department.

```
SELECT d.DEPATMENT_NAME, COUNT (e.DEPATMENT_ID) total_emp_count
FROM EMPLOYEE e, DEPARTMENT d
WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID – this condition pulls the matching employees
GROUP BY e.DEPARTMENT_ID;
```

In the above query instead of count (e.DEPARTMENT_ID), we can give count (1). Both are same.

Some of the Group functions are

Count – it counts the total number of records in the table/s after applying 'where' clause. If where clause is not specified, it gives the total number of records in the table. If Group by clause is applied, it filters the records based on where clause (if any), then groups the records based on the columns in group by clause and gives the total count of records in each grouped category.

SUM – It totals the value in each numeric column. We can use this function to find the total marks of a student, total salary of an employee in a specific period etc.

AVG – It gives the average value of a column, provided column has numeric value. E.g.: Average age of students present in particular class.

MAX – It gives the maximum value in a column. For example, highest scorer in the class can be retrieved by MAX function.

MIN– It gives the minimum value in a column. For example, lowest paid employee in a department can be obtained by MIN.

These group functions can be used with group by clause or without it.

Having Clause – Using this clause we can add conditions to the grouped categories and filter the records. For examples, if we want to display the details of the department which has more than 100 employees.

SELECT d.DEPATMENT_NAME, COUNT (e.DEPATMENT_ID) total_emp_count
FROM EMPLOYEE e, DEPARTMENT d
WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID – this condition pulls the matching employees
GROUP BY e.DEPARTMENT_ID
HAVING COUNT(e.DEPATMENT_ID)>100; -- filters more than 100 employees present in specific department. We cannot give this condition in the where clause as this is the result of Group by clause.

ORDER BY – This clause helps to sort the records that are retrieved. By default, it displays the records in ascending order of primary key. If we need to sort it based on different columns, then we need to specify it in ORDER BY clause. If we need to order by descending order, then DESC keyword has to be added after the column list.

SELECT * FROM EMPLOYEE
ORDER BY EMPLOYEE_NAME DESC; -- Displays all records in descending order of employee name

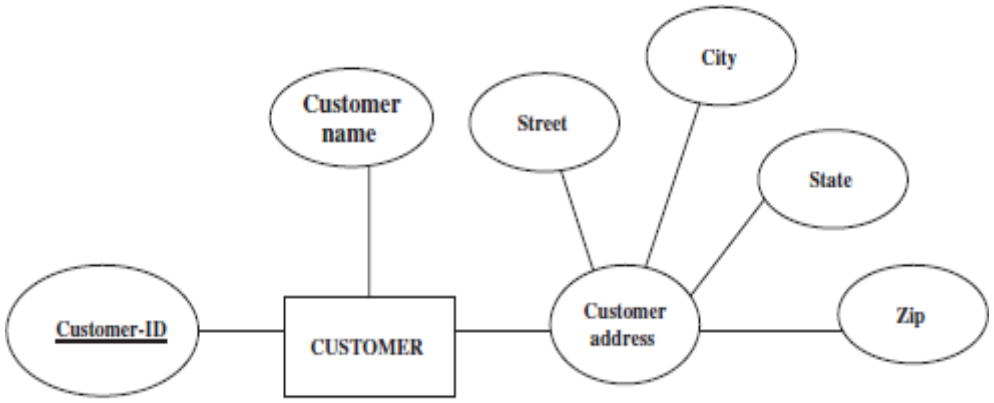
We can combine two or more columns into one column by using || in select statement.
SELECT EMP_FIRST_NAME || ' ' || EMP_LAST_NAME as emp_name
FROM EMPLOYEE WHERE EMP_ID = 1001; -- Here first name and last name of employee with id 1001 to show it as emp_name

Distinct : The SELECT DISTINCT statement is used to return only distinct (different) values.
Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

Syntax : SELECT DISTINCT *column1*, *column2, ...* FROM *table_name*;

LOGICAL DATABASE DESIGN

In this example the composite attribute is the Customer address, which consists of Street, City, State, and Zip.



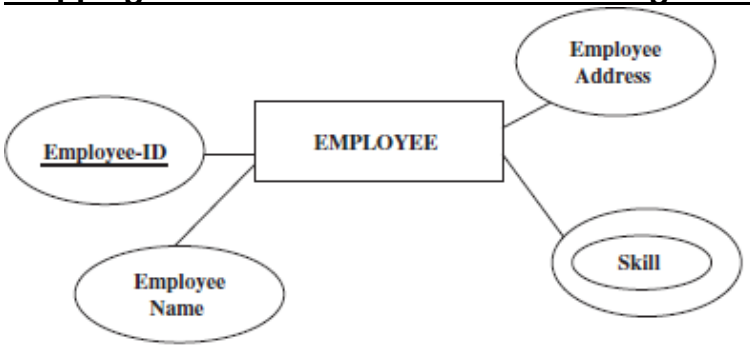
CUSTOMER

Customer-ID	Customer name	Street	City	State	Zip

The first relation contains all of the attributes of the entity type except the multivalued attribute.

The second relation contains two attributes that form the primary key of the second relation. The first of these attributes is the primary key from the first relation, which becomes a foreign key in the second relation. The second is the multivalued attribute.

Mapping Multivalued Attributes in ER Diagram to Tables:



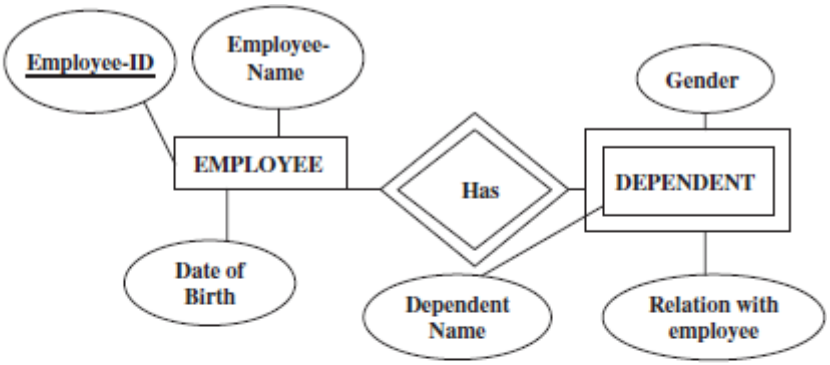
EMPLOYEE

Employee-ID	Employee-Name	Employee-Address

EMPLOYEE-SKILL

EMPLOYEE-ID	Skill

Converting "Weak Entities" in ER Diagram to Tables



The corresponding table is given by

EMPLOYEE

Employee-ID	Employee-Name	Date of Birth

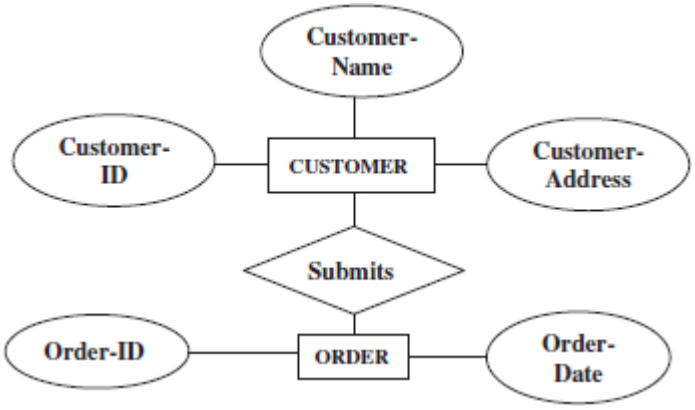
DEPENDENT

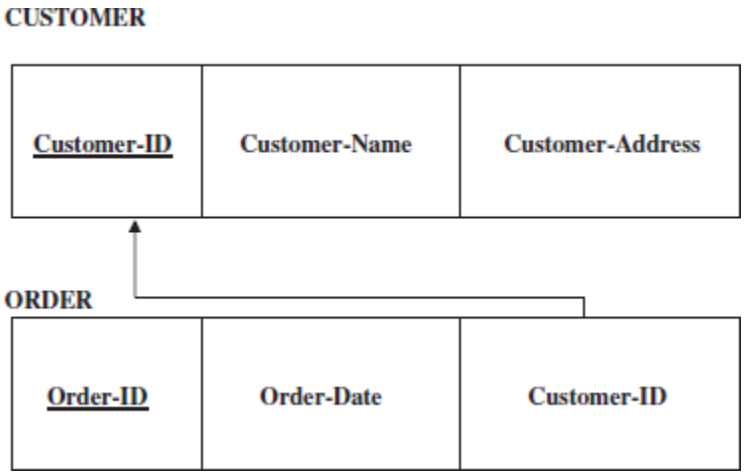
Dependent-Name	Gender	Employee-ID	Relation with Employee

Converting Binary Relationship to Table

A relationship which involves two entities can be termed as binary relationship. This binary relationship can be one-to-one, one-to-many, many-to-one and many-to-many.

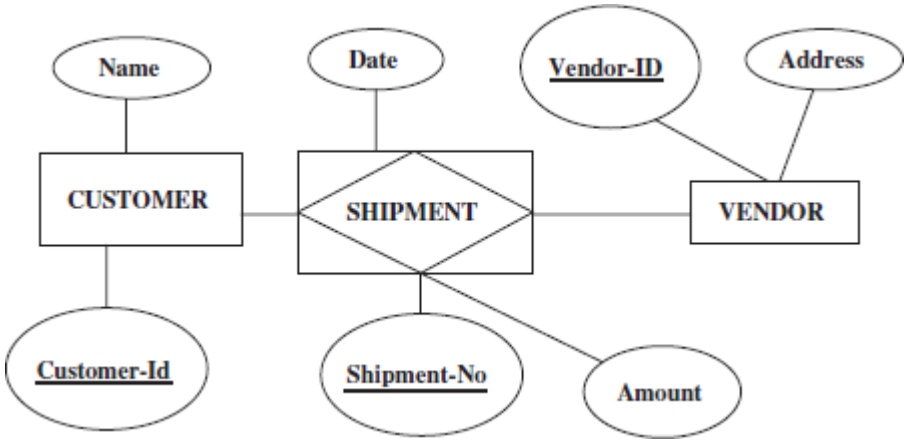
Mapping one-to-Many Relationship

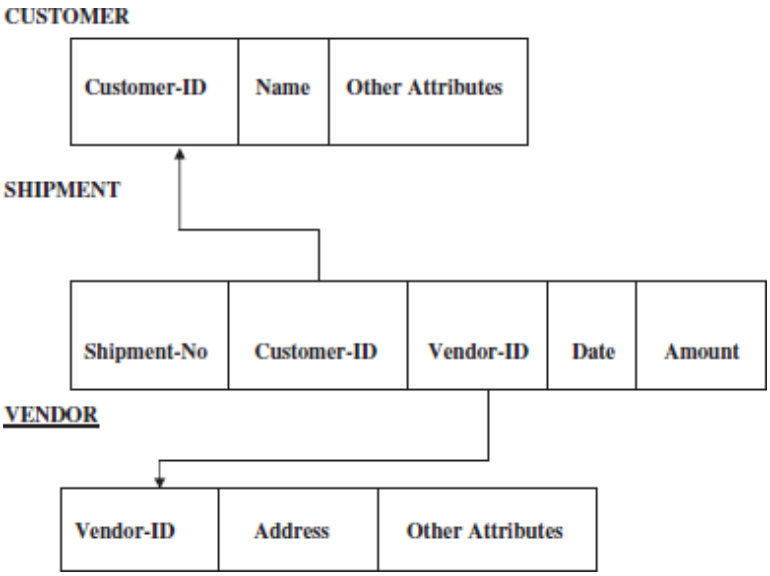




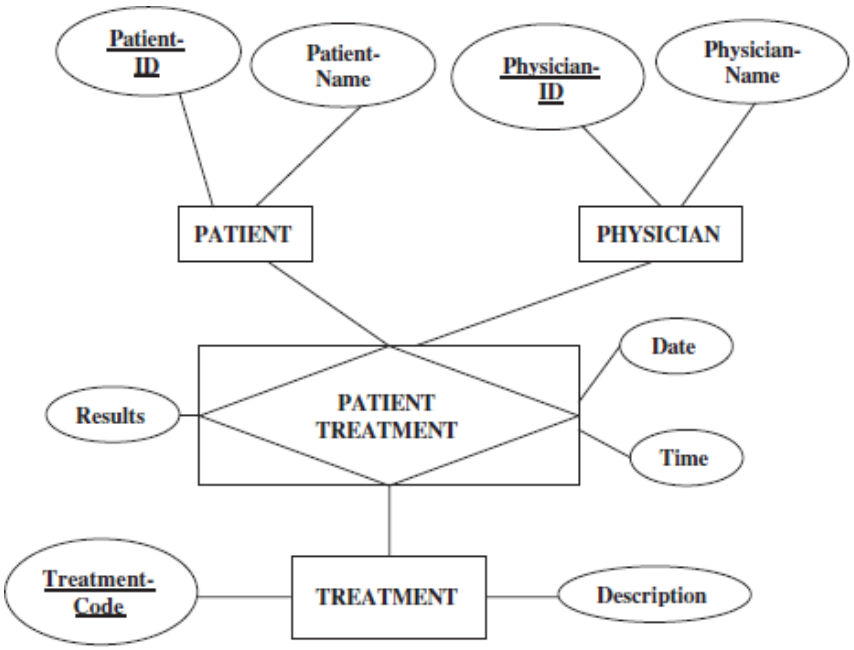
Mapping Associative Entity to Tables

Many-to-many relationship can be modeled as an associative entity in the ER diagram.





Converting Ternary Relationship to Tables



PATIENT TREATMENT

Patient-ID	Patient-Name

PHYSICIAN

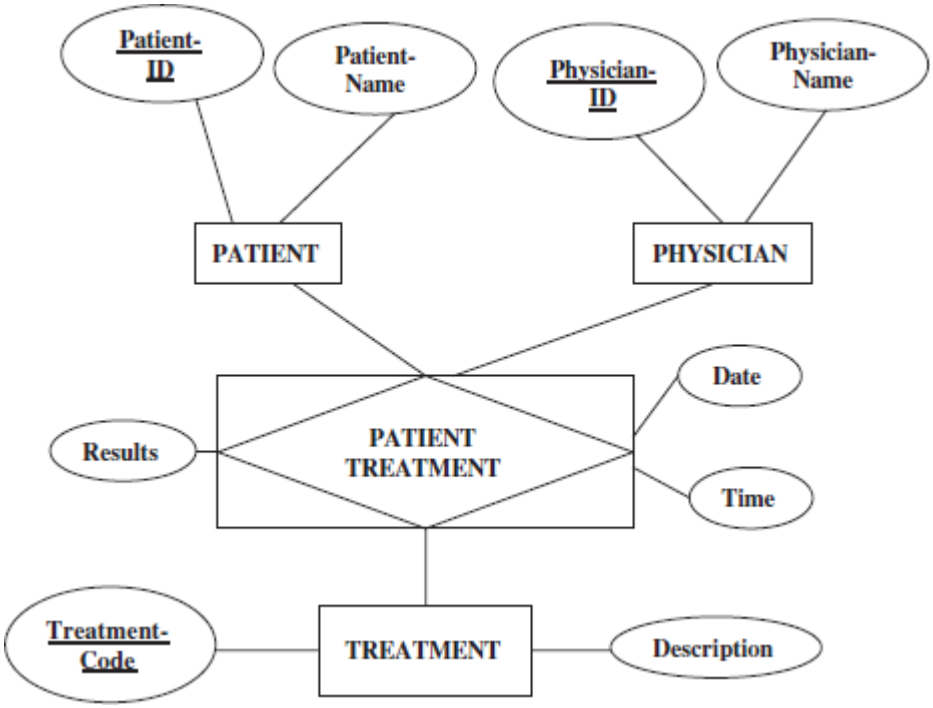
Physician-ID	Physician-Name

PATIENT TREATMENT

Patient-ID	Physician-ID	Treatment-Code	Date	Time	Results

TREATMENT

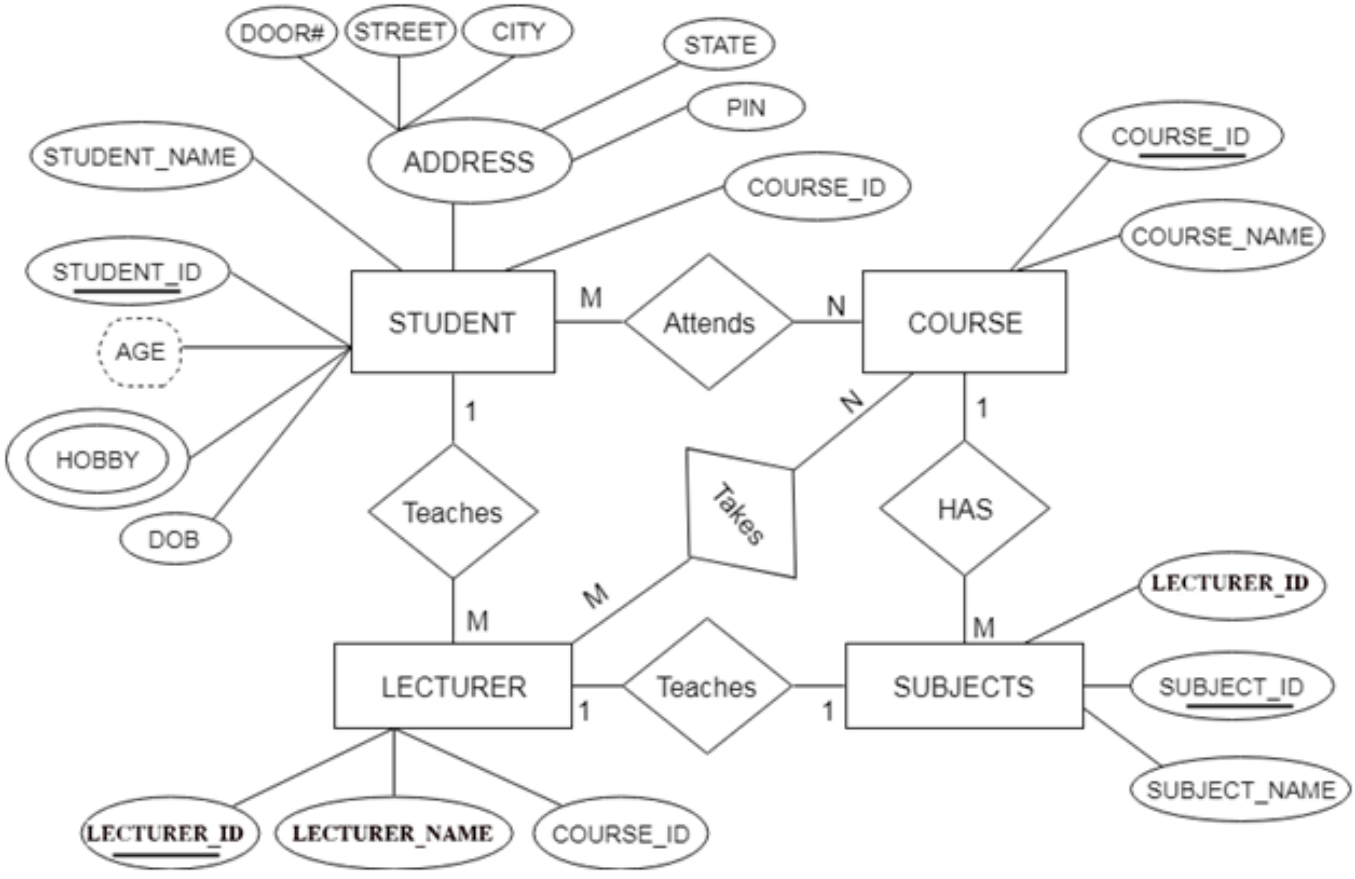
Treatment-Code	Description



The database can be represented using the notations, and these notations can be reduced to a collection of tables.

In the database, every entity set or relationship set can be represented in tabular form.

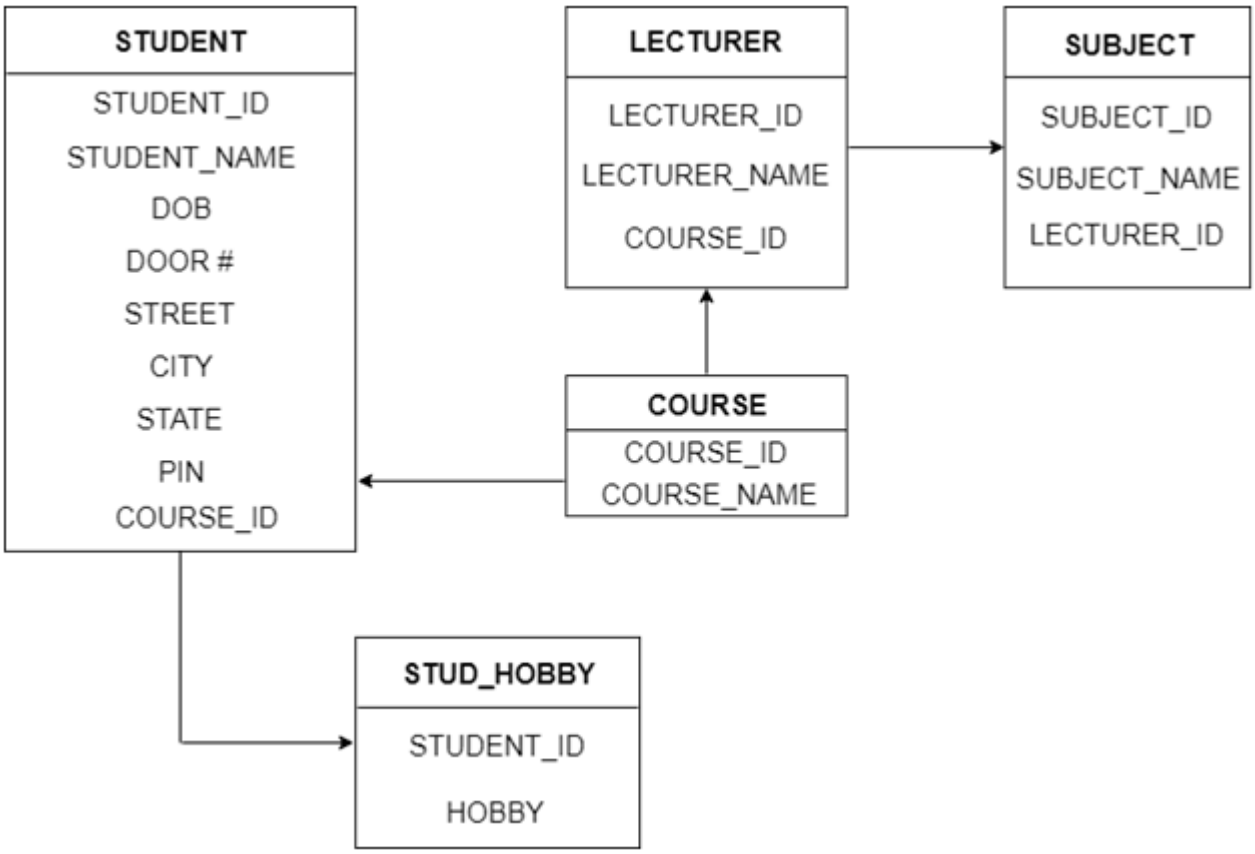
The ER diagram is given below:



There are some points for converting the ER diagram to the table:

- **Entity type becomes a table.** In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.
- **All single-valued attribute becomes a column for the table.** In the STUDENT entity, STUDENT_NAME and STUDENT_ID form the column of STUDENT table. Similarly, COURSE_NAME and COURSE_ID form the column of COURSE table and so on.
- **A key attribute of the entity type represented by the primary key.** In the given ER diagram, COURSE_ID, STUDENT_ID, SUBJECT_ID, and LECTURE_ID are the key attribute of the entity.
- **The multivalued attribute is represented by a separate table.** In the student table, a hobby is a multivalued attribute. So it is not possible to represent multiple values in a single column of STUDENT table. Hence we create a table STUD_HOBBY with column name STUDENT_ID and HOBBY. Using both the column, we create a composite key.
- **Composite attribute represented by components.** In the given ER diagram, student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET, and STATE. In the STUDENT table, these attributes can merge as an individual column.
- **Derived attributes are not considered in the table.** In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.

Using these rules, you can convert the ER diagram to tables and columns and assign the mapping between the tables. Table structure for the given ER diagram is as below:



INTRODUCTION TO VIEWS

As you can see from the above image, we can extract data columns from more than one table by running queries on the data tables. Views contain only the definition of the view data in the data dictionary, not the actual data.

The view has two primary purposes:

- Simplifying complex SQL queries.
- Restricting users from accessing sensitive data.

Sample Table : For creating a view (simple and complex views), updating and deleting the views we will take some sample data and tables that store the data.

Let's take an employee table that store data of employees in a particular company:

Employee Table:

This table contains details of employees in a particular company and has data fields such as EmpID, EmpName, Address, Contact. We have added 6 records of employees for our purpose.

EmpID	EmpName	Address	Contact
1	Alex	London	05-12421969
2	Adolf	San Francisco	01-12584365
3	Aryan	Delhi	91-9672367287
4	Bhuvan	Hyderabad	91-9983288383
5	Carol	New York	01-18928992
6	Steve	California	01-13783282

EmpRole Table: This table contains details of employees' roles in a particular company and has data fields as EmpID, Role, Dept. We have stored all the records particular to the EmpID of each employee.

EmpID	Role	Dept
1	Intern	Engineering
2	Trainee	IT
3	Executive	HR
4	SDE-1	Engineering
5	SDE-2	Engineering
6	Technical Architect	Engineering

Creating View : **The view can be created by using the CREATE VIEW statement, Views can be simple or complex depending on their usage.**

Syntax: CREATE VIEW veiwName AS SELECT column1, column2,.... FROM tableName WHERE condition;

Here, viewName is the Name for the View we set, tableName is the Name of the table and condition is the Condition by which we select rows.

Creating a Simple View : Simple view is the view that is made from a single table, It takes only one table and just the conditions, It also does not take any inbuilt SQL functions like AVG(), MIN(), MAX() etc, or GROUP BY clause.

While creating a simple view, we are not creating an actual table, we are just projecting the data from the original table to create a view table.

Let's look at some examples for creating a simple view.

Example 1: In this example, we are creating a view table from Employee Table for getting the EmpID and EmpName. So the query will be:

```
CREATE VIEW EmpView1 AS  SELECT EmpID, EmpName   FROM Employee;
```

Now to see the data in the EmpView1 view created by us, We have to simply use the SELECT statement.

```
SELECT * from EmpView1;
```

Output:

The view table EmpView1 that we have created from Employee Table contains EmpID and EmpName as its data fields.

EmpID	EmpName
1	Alex
2	Adolf
3	Aryan
4	Bhuvan
5	Carol
6	Steve

Example 2: In this example, we are creating a view table from EmpRole Table for getting the EmpID and Dept for employees having EmpIDs less than 4. So the query will be:

```
CREATE VIEW EmpView2 AS SELECT EmpID, Dept  FROM EmpRole   WHERE EmpID<4;
```

Now to see the data in the EmpView2 view created by us, We have to simply use the SELECT statement.

```
SELECT * from EmpView2;
```

Output:

The view table EmpView2 that we have created from EmpRole Table contains EmpID and Dept as its data fields.

EmpID	Dept
1	Engineering
2	IT
3	HR

DESTROYING/ALTERING TABLES AND VIEWS

ALTERING TABLE : Alter command is used to modify the existing database objects. It can add, delete/drop or modify columns in the existing table. It can also be used to add and drop various constraints on the existing table.

ALTER TABLE Employees ADD Salary **DOUBLE**(8,2);
This query will add a column Salary to the existing table Employees. If the table Employees **do** not exist, an error will be generated.

ALTER TABLE Employees DROP COLUMN Salary;
This query will delete the column Salary from the table Employees.

ALTER TABLE Employees MODIFY Name **VARCHAR**(40);
This query will modify the existing column Name in table Employees and change the size to 40 from 20.

DESTROYING TABLE: DROP command is used to delete the various existing database objects. It deletes an entire database, an entire table, a view of a table or other objects in the database. If you **drop** a table, all the rows in the table are deleted and the table structure is removed from the database. Once a table is **dropped** we cannot get it back.

DROP TABLE Employees;
This SQL command will remove the table structure along with its data from the database.

Deleting view

what if we don't need our created views anymore, So we need to delete the view so as we DROP a table in SQL, similarly, we can delete or drop a view using the DROP statement. The DROP statement completely deletes the structure of the view.

Syntax:
DROP VIEW ViewName; ------ Here ViewName is the name of the view to be deleted.

Example:
Let's delete one of our created views, say EmpView1.
DROP VIEW EmpView1;
Let's use the SELECT statement on Deleted View.
SELECT * from EmpView1;

Output:
Now SQL Editor will give us an error saying the table does not exist as the table is now been deleted.
ORA-00942: table or view does not exist

Updating View

Suppose we want to add more columns to the created view so we will have to update the view.

For updating the views we can use CREATE OR REPLACE VIEW statement, new columns will replace or get added to the view.

Syntax:

```
CREATE OR REPLACE VIEW ViewName AS
SELECT column1,coulmn2,..
FROM TableName
WHERE condition;
```

For Updating a view CREATE OR REPLACE statement is used. Here, viewName is the Name of the view we want to update, tableName is the Name of the table and condition is the Condition by which we select rows.

Example 1:

let's take the above created simple view EmpView1 and we want to one more column of address to our EmpView1 from Employee Table.

```
CREATE OR REPLACE VIEW EmpView1 AS
SELECT EmpID, EmpName, Address
FROM Employee;
```

Now to see the data in the EmpView1 view created by us, We have to simply use the SELECT statement.

```
SELECT * from EmpView1;
```

Output:

The data fields of view table EmpView1 have been updated to EmpID, EmpName, and Address.

EmpID	EmpName	Address
1	Alex	London
2	Adolf	San Francisco
3	Aryan	Delhi
4	Bhuvan	Hyderabad
5	Carol	New York
6	Steve	California

Relational Algebra: Preliminaries, Relational Algebra operators.

RELATIONAL ALGEBRA

Introduction

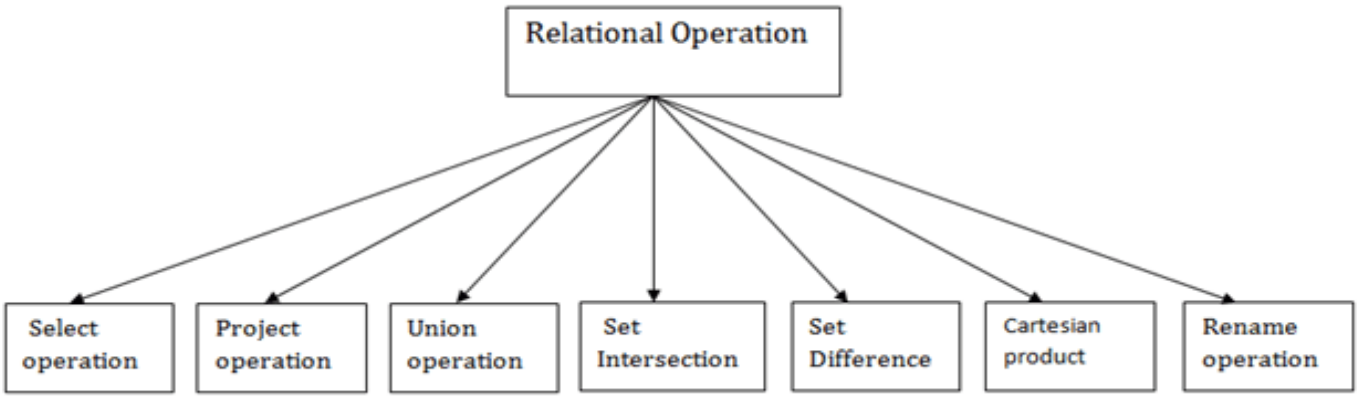
1. The relational algebra is a notional language with operations
2. That work on one or more relations to define another relation without changing the original relation.
3. The both the operands and the results are relations;
4. The output from one operation can become the input to another operation.
5. This allows expressions to be nested in the relational algebra. This property is called closure.
6. Relational algebra is an abstract language,
7. It means that the queries formulated in relational algebra are not intended to be executed on a computer.
8. Relational algebra consists of group of relational operators that can be used to manipulate relations to obtain a desired result.
9. Knowledge about relational algebra allows us to understand query execution and optimization in relational database management system.

Relational Algebra Operations (Relational Set operators)

Operations in relational algebra can be classified into set operation and database operations.

1. Unary and Binary Operations

1. Unary operation involves one operand,
   a. selection
   b. Projection
2. Binary operation involves two operands.
   a. Union operation
   b. Difference operation
   c. Cartesian product operation
   d. Intersection operation
   e. Division operation
   f. Join operations operation



1. Select Operation:

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

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: 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 ∏.

Notation: ∏ A1, A2, An (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: ∏ 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 ∪.

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: ∏ 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 ∩.

Notation: R ∩ S

Example: Using the above DEPOSITOR table and BORROW table

Input: ∏ 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 (-).

Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table

Input: ∏ 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.

  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: EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

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. ρ(STUDENT1, STUDENT)

b. Join Operations

1. Join operation combines two relations to create a new relation.
2. The tables should be joined based on a common column.
3. The common column should be compatible in terms of domain.
4. It allows information to be combined from two or more tables.
5. Join is the real power behind the relation database, allowing the use of independent table linked by common attributes (Characteristics).

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

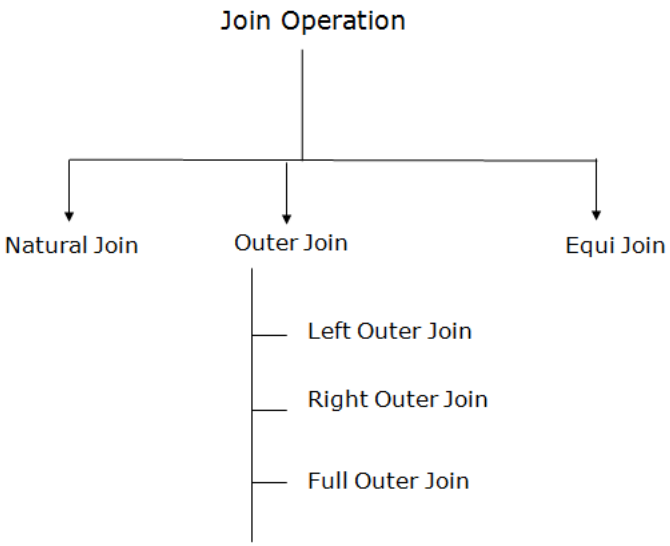
EMP_CODE	SALARY
101	50000
102	30000
103	25000

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:∏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:(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: 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: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: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: CUSTOMER ⋈ PRODUCT

Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida