## Unit-1

**1. Database Management System: Database:** Database is a collection of inter-related data which helps in efficient retrieval, insertion and deletion of data from database and organizes the data in the form of tables, views, schemas, reports etc. For Example, university database organizes the data about students, faculty, and admin staff etc. which helps in efficient retrieval, insertion and deletion of data from it.

**(i).DDL** is short name of Data Definition Language, which deals with database schemas and descriptions, of how the data should reside in the database.

- CREATE: to create a database and its objects like (table, index, views, store procedure, function, and triggers)
- ALTER: alters the structure of the existing database
- DROP: delete objects from the database
- TRUNCATE: remove all records from a table, including all spaces allocated for the records are removed
- COMMENT: add comments to the data dictionary
- RENAME: rename an object.

**(ii)DML** is short name of Data Manipulation Language which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.

- SELECT: retrieve data from a database
- INSERT: insert data into a table
- UPDATE: updates existing data within a table
- DELETE: Delete all records from a database table
- MERGE: UPSERT operation (inserts or update)
- CALL: call a PL/SQL or Java subprogram
- EXPLAIN PLAN: interpretation of the data access path
- LOCK TABLE: concurrency Control

**(iii)Database Management System:** The software which is used to manage database is called Database Management System (DBMS). For Example, MySQL, Oracle etc. are popular commercial DBMS used in different applications. DBMS allows users the following tasks:

**Data Definition:** It helps in creation, modification and removal of definitions that define the organization of data in database.
**Data Updating:** It helps in insertion, modification and deletion of the actual data in the database.
**Data Retrieval:** It helps in retrieval of data from the database which can be used by applications for various purposes.
**User Administration:** It helps in registering and monitoring users, enforcing data security, monitoring performance, maintaining data integrity, dealing with concurrency control and recovering information corrupted by unexpected failure.

**Paradigm Shift from File System to DBMS**

File System manages data using files in hard disk. Users are allowed to create, delete, and update the files according to their requirement. Let us consider the example of file based University Management System. Data of students is available to their respective Departments, Academics Section, Result Section, Accounts Section, Hostel Office etc.

Some of the data is common for all sections like Roll No, Name, Father Name, Address and Phone number of students but some data is available to a particular section only like Hostel allotment number which is a part of hostel office. Let us discuss the issues with this system:

- **Redundancy of data:** Data is said to be redundant if same data is copied at many places. If a student wants to change Phone number, he has to get it updated at various sections. Similarly, old records must be deleted from all sections representing that student.
- **Inconsistency of Data:** Data is said to be inconsistent if multiple copies of same data does not match with each other. If Phone number is different in Accounts Section and Academics Section, it will be inconsistent. Inconsistency may be because of typing errors or not updating all copies of same data.
- **Difficult Data Access:** A user should know the exact location of file to access data, so the process is very cumbersome and tedious. If user wants to search student hostel allotment number of a student from 10000 unsorted students' records, how difficult it can be.
- **Unauthorized Access:** File System may lead to unauthorized access to data. If a student gets access to file having his marks, he can change it in unauthorized way.
- **No Concurrent Access:** The access of same data by multiple users at same time is known as concurrency. File system does not allow concurrency as data can be accessed by only one user at a time.
- **No Backup and Recovery:** File system does not incorporate any backup and recovery of data if a file is lost or corrupted.

**Purpose of Database Management Systems**

Organizations use large amounts of data. A **database management system** (DBMS) is a software tool that makes it possible to organize data in a database.

The standard acronym for database management system is **DBMS**, so you will often see this instead of the full name. The ultimate purpose of a database management system is to store and transform data into information to support making decisions.

A DBMS consists of the following three elements:

1. The **physical database**: the collection of files that contain the data
2. The **database engine**: the software that makes it possible to access and modify the contents of the database
3. The **database scheme**: the specification of the logical structure of the data stored in the database

Functions of a DBMS:So, what does a DBMS really do? It organizes your files to give you more control over your data.

A DBMS makes it possible for users to create, edit and update data in database files. Once created, the DBMS makes it possible to store and retrieve data from those database files.

More specifically, a DBMS provides the following functions:

- Concurrency: concurrent access (meaning 'at the same time') to the same database by multiple users
- Security: security rules to determine access rights of users

- Backup and recovery: processes to back-up the data regularly and recover data if a problem occurs
- Integrity: database structure and rules improve the integrity of the data
- Data descriptions: a data dictionary provides a description of the data

Within an organization, the development of the database is typically controlled by **database administrators (DBAs)** and other specialists. This ensures the database structure is efficient and reliable.

**2. 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.
- **Sales**: To store customer information, production information and invoice details.
- **Airlines**: To travel though airlines, we make early reservations, this reservation information along with flight schedule is stored in database.
- **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 hell lot 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.

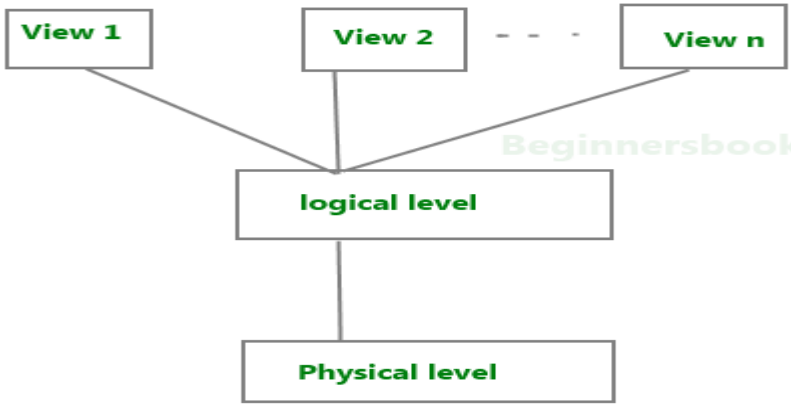
I have mentioned very few applications, this list is never going to end if we start mentioning all the DBMS applications.

**3. View of Data in DBMS:**

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. In the previous tutorial, we discussed the three level of DBMS architecture, The top level of that architecture is “view level”. 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.

To fully understand the view of data, you must have a basic knowledge of data abstraction and instance & schema. Refer these two tutorials to learn them in detail.

1. Data abstraction
2. Instance and schema
3. 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.



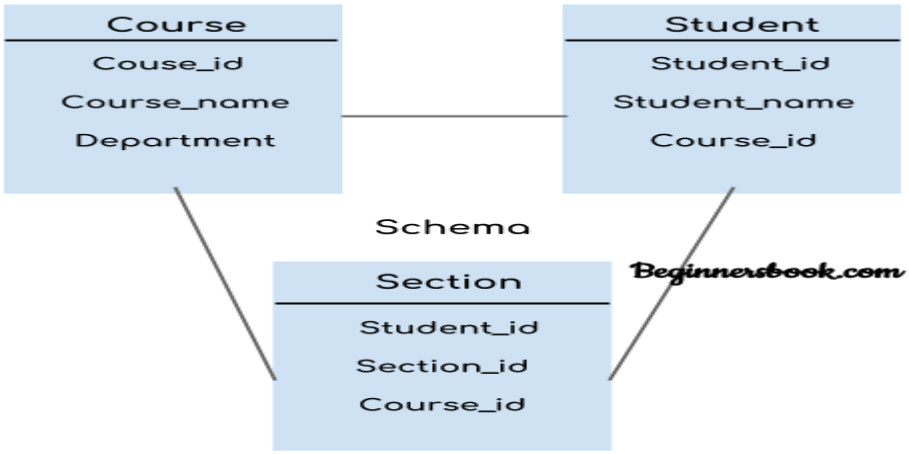

Three Levels of data abstraction

4. **We have three levels of abstraction**:
   **Physical level**: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.
5. **Logical level**: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.
6. **View level**: Highest level of data abstraction. This level describes the user interaction with database system.
7. **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.
8. 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.
9. 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.

**4. Instance and schema in DBMS: Definition of schema**: Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema.

For example: 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.



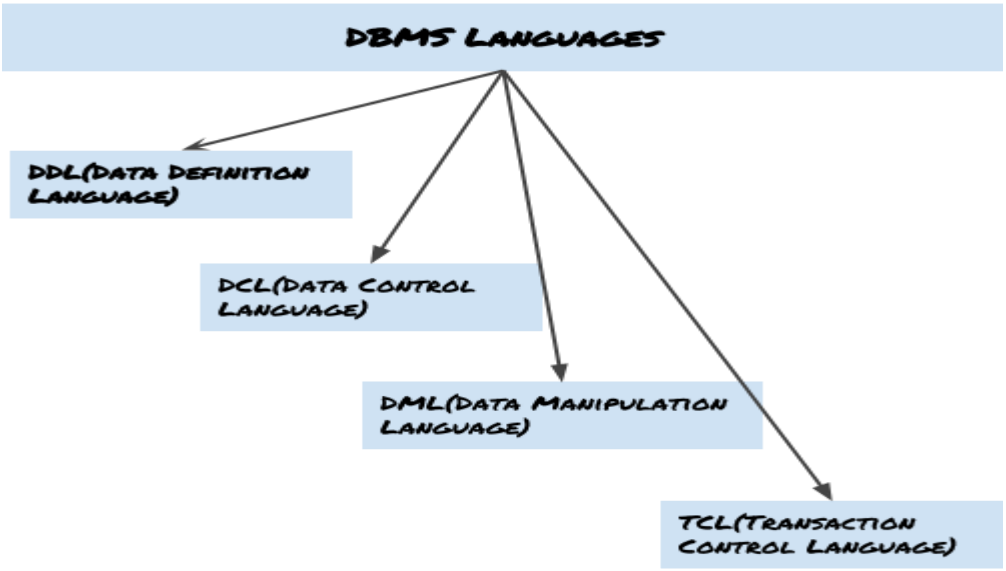

The design of a database at physical level is called **physical schema**, how the data stored in blocks of storage is described at this level.

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.

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

**Types of DBMS languages:**




**Data Definition Language (DDL)**

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

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

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

**Data Manipulation Language (DML)**

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

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

**Data Control language (DCL)**

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

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

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

**Transaction Control Language(TCL)**

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

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

**6. Data models in DBMS**

**Data Model** is a logical structure of Database. It describes the design of database to reflect entities, attributes, relationship among data, constrains etc.

Types of Data Models

There are several types of data models in DBMS. We will cover them in detail in separate articles(Links to those separate tutorials are already provided below). In this guide, we will just see a basic overview of types of models.

**Object based logical Models** – Describe data at the conceptual and view levels.

1. E-R Model
2. Object oriented Model

**Record based logical Models** – Like Object based model, they also describe data at the conceptual and view levels. These models specify logical structure of database with records, fields and attributes.

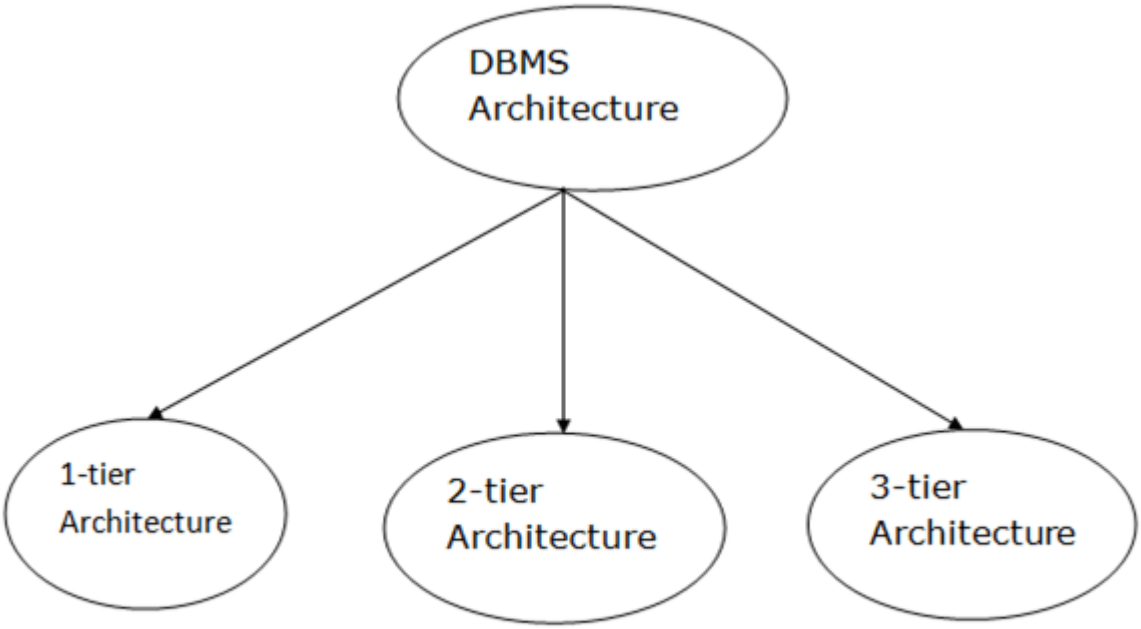
1. Relational Model
2. Hierarchical Model
3. Network Model – Network Model is same as hierarchical model except that it has graph-like structure rather than a tree-based structure. Unlike hierarchical model, this model allows each record to have more than one parent record.

**Physical Data Models** – These models describe data at the lowest level of abstraction.

## 7.DBMS Architecture

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

### Types of DBMS Architecture




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

### 1-Tier Architecture

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

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

## 2-Tier Architecture

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

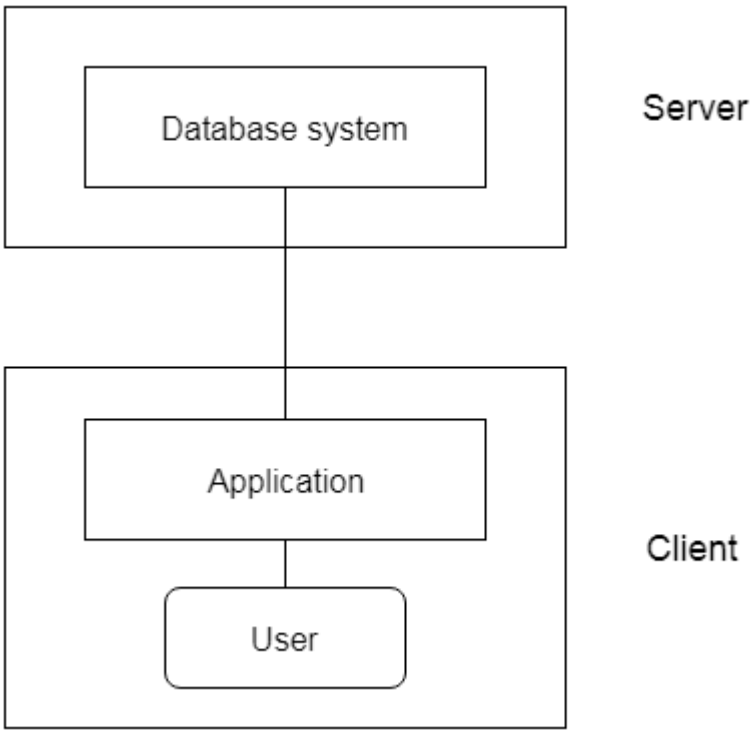



**Fig: 2-tier Architecture**

## 3-Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.

- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.

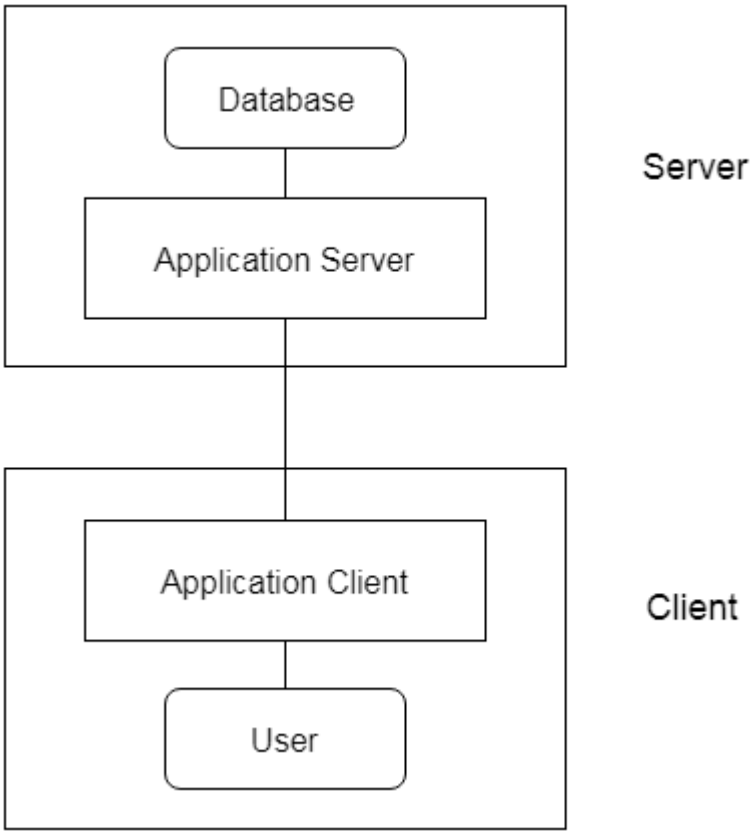
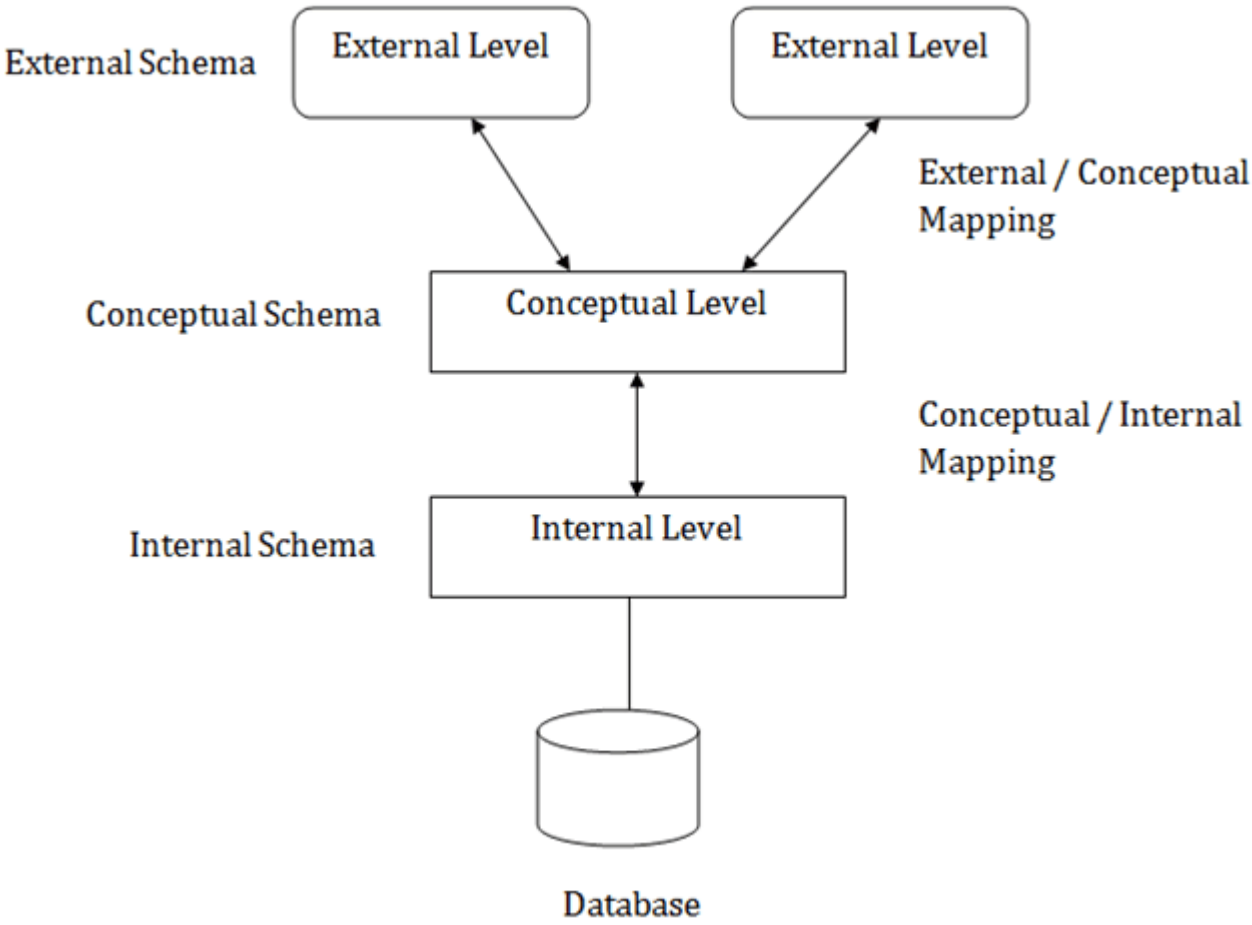



**Fig: 3-tier Architecture**

## Three schema Architecture

- The three schema architecture is also called ANSI/SPARC architecture or three-level architecture.
- This framework is used to describe the structure of a specific database system.
- The three schema architecture is also used to separate the user applications and physical database.
- The three schema architecture contains three-levels. It breaks the database down into three different categories.

**The three-schema architecture is as follows:**




**In the above diagram:**

- It shows the DBMS architecture.
- Mapping is used to transform the request and response between various database levels of architecture.
- Mapping is not good for small DBMS because it takes more time.
- In External / Conceptual mapping, it is necessary to transform the request from external level to conceptual schema.
- In Conceptual / Internal mapping, DBMS transform the request from the conceptual to internal level.

1. Internal Level

- The internal level has an internal schema which describes the physical storage structure of the database.
- The internal schema is also known as a physical schema.

- It uses the physical data model. It is used to define that how the data will be stored in a block.
- The physical level is used to describe complex low-level data structures in detail.

### 2. Conceptual Level

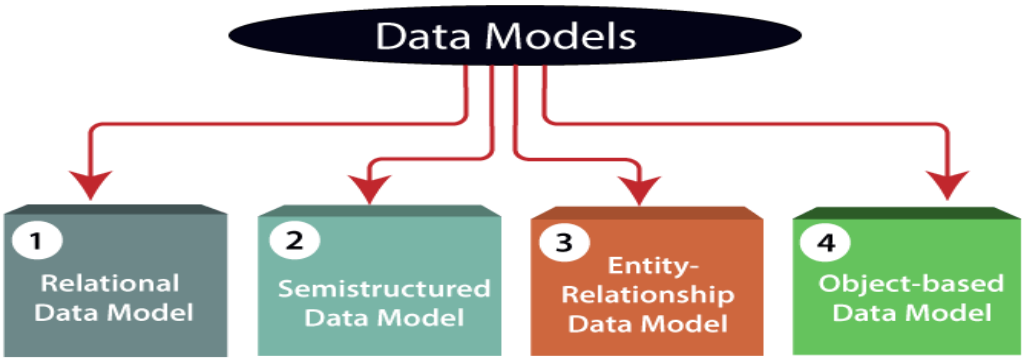
- The conceptual schema describes the design of a database at the conceptual level. Conceptual level is also known as logical level.
- The conceptual schema describes the structure of the whole database.
- The conceptual level describes what data are to be stored in the database and also describes what relationship exists among those data.
- In the conceptual level, internal details such as an implementation of the data structure are hidden.
- Programmers and database administrators work at this level.

### 3. External Level

- At the external level, a database contains several schemas that sometimes called as subschema. The subschema is used to describe the different view of the database.
- An external schema is also known as view schema.
- Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.
- The view schema describes the end user interaction with database systems.

## 8.Data Models

Data Model is the modeling of the data description, data semantics, and consistency constraints of the data. It provides the conceptual tools for describing the design of a database at each level of data abstraction. Therefore, there are following four data models used for understanding the structure of the database:



**1) Relational Data Model:** This type of model designs the data in the form of rows and columns within a table. Thus, a relational model uses tables for representing data and in-between relationships. Tables are also called relations. This model was initially described by Edgar F. Codd, in 1969. The relational data model is the widely used model which is primarily used by commercial data processing applications.

**2) Entity-Relationship Data Model:** An ER model is the logical representation of data as objects and relationships among them. These objects are known as entities, and relationship is an association among these entities. This model was designed by Peter Chen and published in 1976 papers. It was widely used in database designing. A set of attributes describe the entities. For example, student_name, student_id describes the 'student' entity. A set of the same type of entities is known as an 'Entity set', and the set of the same type of relationships is known as 'relationship set'.

**3) Object-based Data Model:** An extension of the ER model with notions of functions, encapsulation, and object identity, as well. This model supports a rich type system that includes structured and collection types. Thus, in 1980s, various database systems following the object-oriented approach were developed. Here, the objects are nothing but the data carrying its properties.

**4) Semistructured Data Model:** This type of data model is different from the other three data models (explained above). The semistructured data model allows the data specifications at places where the individual data items of the same type may have different attributes sets. The Extensible Markup Language, also known as XML, is widely used for representing the semistructured data. Although XML was initially designed for including the markup information to the text document, it gains importance because of its application in the exchange of data.

**Relational databases:** RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

What is a table?

The data in an RDBMS is stored in database objects which are called as **tables**. This table is basically a collection of related data entries and it consists of numerous columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database. The following program is an example of a CUSTOMERS table −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
```

```
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

What is a field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

What is a Record or a Row?

A record is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table −

```
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
```

A record is a horizontal entity in a table.

What is a column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would be as shown below −

```
+-----------+
| ADDRESS   |
+-----------+
| Ahmedabad |
| Delhi     |
| Kota      |
| Mumbai    |
| Bhopal    |
| MP        |
| Indore    |
+----+------+
```

What is a NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a record creation.

SQL Constraints

Constraints are the rules enforced on data columns on a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints can either be column level or table level. Column level constraints are applied only to one column whereas, table level constraints are applied to the entire table.

Following are some of the most commonly used constraints available in SQL −

- NOT NULL Constraint − Ensures that a column cannot have a NULL value.
- DEFAULT Constraint − Provides a default value for a column when none is specified.
- UNIQUE Constraint − Ensures that all the values in a column are different.
- PRIMARY Key − Uniquely identifies each row/record in a database table.
- FOREIGN Key − Uniquely identifies a row/record in any another database table.
- CHECK Constraint − The CHECK constraint ensures that all values in a column satisfy certain conditions.
- INDEX − Used to create and retrieve data from the database very quickly.

Data Integrity

The following categories of data integrity exist with each RDBMS −

- **Entity Integrity −** There are no duplicate rows in a table.
- **Domain Integrity −** Enforces valid entries for a given column by restricting the type, the format, or the range of values.
- **Referential integrity −** Rows cannot be deleted, which are used by other records.
- **User-Defined Integrity −** Enforces some specific business rules that do not fall into entity, domain or referential integrity.

Database Normalization

Database normalization is the process of efficiently organizing data in a database. There are two reasons of this normalization process −

- Eliminating redundant data, for example, storing the same data in more than one table.
- Ensuring data dependencies make sense.

Both these reasons are worthy goals as they reduce the amount of space a database consumes and ensures that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into normal forms; think of a form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure, so that it complies with the rules of first normal form, then second normal form and finally the third normal form.

It is your choice to take it further and go to the fourth normal form, fifth normal form and so on, but in general, the third normal form is more than enough.

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

**What is 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 designing 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 design of the database onto physical media using hardware resources and software systems such as database management systems (DBMS).

**Why Database Design is Important ?**

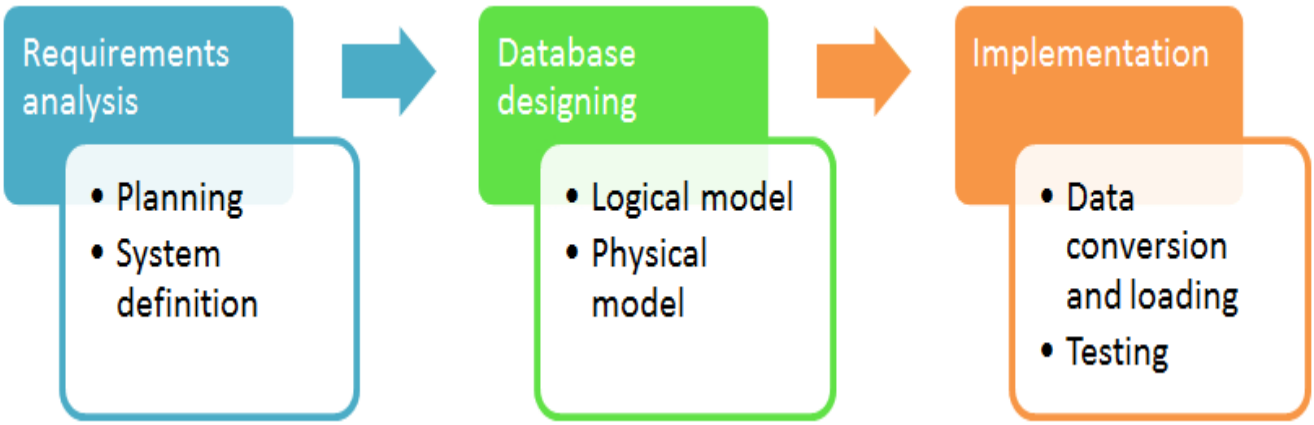
It helps produce database systems

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

Database designing is crucial to **high performance** database system.

Note , the genius of a database is in its design . Data operations using SQL is relatively simple

**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 necessary have to be followed religiously in a sequential manner.

On small database systems, the database system development life cycle 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.

**Requirements analysis**

- **Planning** - This stages concerns 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 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**

**What is Normalization?**

**NORMALIZATION** is a database design technique that organizes tables in a manner that reduces redundancy and dependency of data. Normalization divides larger tables into smaller tables and links them using relationships. The purpose of Normalization is to eliminate redundant (useless) data and ensure data is stored logically.

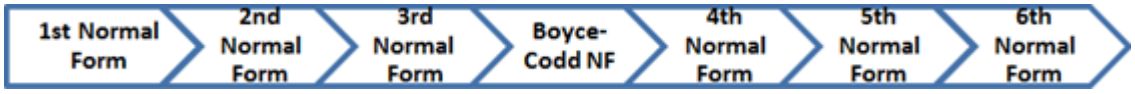
The inventor of the relational model Edgar Codd proposed the theory of normalization with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

In this tutorial, you will learn-

- 1NF Rules
- 2NF Rules
- 3NF Rules
- BCNF (Boyce-Codd Normal Form)

**Database Normal Forms**

The Theory of Data Normalization in SQL is still being developed further. For example, there are discussions even on 6th Normal Form. **However, in most practical applications, normalization achieves its best in 3rd Normal Form**. The evolution of Normalization theories is illustrated below-




**Database Normalization Example:**

We will study normalization with the help of a case study. Assume, a video library maintains a database of movies rented out. Without any normalization, all information is stored in one table as shown below.

| FULL NAMES | PHYSICAL ADDRESS | MOVIES RENTED | SALUTATION |
|---|---|---|---|
| Janet Jones | First Street Plot No 4 | Pirates of the Caribbean, Clash of the Titans | Ms. |
| Robert Phil | 3rd Street 34 | Forgetting Sarah Marshal, Daddy's Little Girls | Mr. |
| Robert Phil | 5th Avenue | Clash of the Titans | Mr. |

Here you see **Movies Rented column has multiple values.** Now let's move into 1st Normal Forms:

1NF (First Normal Form) Rules

- Each table cell should contain a single value.
- Each record needs to be unique.

The above table in 1NF-

**1NF Example**

| FULL NAMES | PHYSICAL ADDRESS | MOVIES RENTED | SALUTATION |
|---|---|---|---|
| Janet Jones | First Street Plot No 4 | Pirates of the Caribbean | Ms. |
| Janet Jones | First Street Plot No 4 | Clash of the Titans | Ms. |
| Robert Phil | 3rd Street 34 | Forgetting Sarah Marshal | Mr. |
| Robert Phil | 3rd Street 34 | Daddy's Little Girls | Mr. |
| Robert Phil | 5th Avenue | Clash of the Titans | Mr. |

Before we proceed let's understand a few things --

**What is a KEY?**

A KEY is a value used to identify a record in a table uniquely. A KEY could be a single column or combination of multiple columns

Note: Columns in a table that are NOT used to identify a record uniquely are called non-key columns.

***What is a Primary Key?***

***A primary is a single column value used to identify a database record uniquely.***

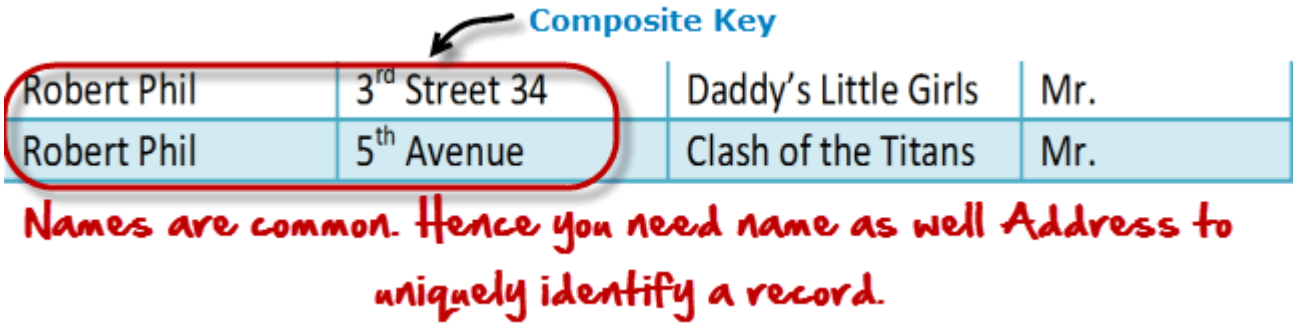
It has following attributes

- A primary key cannot be NULL
- A primary key value must be unique
- The primary key values should rarely be changed
- The primary key must be given a value when a new record is inserted.

**What is Composite Key?**

A composite key is a primary key composed of multiple columns used to identify a record uniquely

In our database, we have two people with the same name Robert Phil, but they live in different places.




Hence, we require both Full Name and Address to identify a record uniquely. That is a composite key.

Let's move into second normal form 2NF

**2NF (Second Normal Form) Rules**

- Rule 1- Be in 1NF
- Rule 2- Single Column Primary Key

It is clear that we can't move forward to make our simple database in 2nd Normalization form unless we partition the table above.

| MEMBERSHIP ID | MOVIES RENTED |
|---|---|
| 1 | Pirates of the Caribbean |
| 1 | Clash of the Titans |
| 2 | Forgetting Sarah Marshal |
| 2 | Daddy's Little Girls |
| 3 | Clash of the Titans |

We have divided our 1NF table into two tables viz. Table 1 and Table2. Table 1 contains member information. Table 2 contains information on movies rented.

We have introduced a new column called Membership_id which is the primary key for table 1. Records can be uniquely identified in Table 1 using membership id

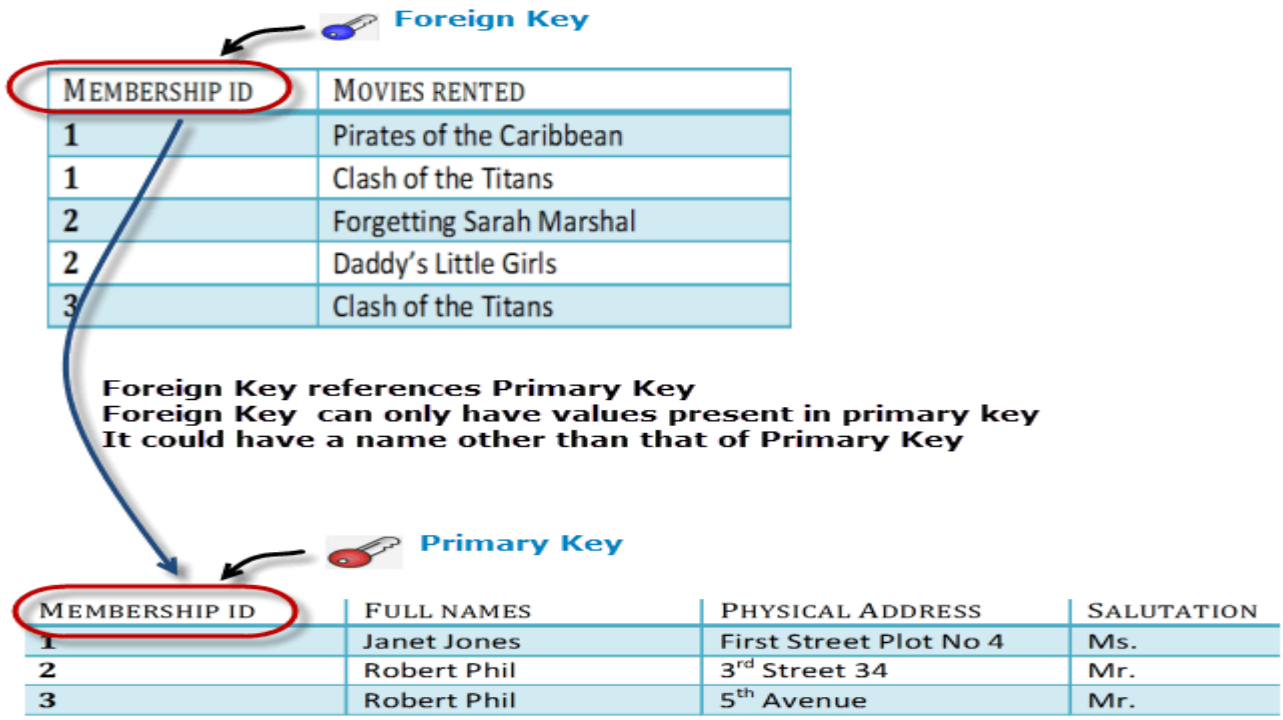
**Database - Foreign Key**

In Table 2, Membership_ID is the Foreign Key

| MEMBERSHIP ID | MOVIES RENTED |
|---|---|
| 1 | Pirates of the Caribbean |
| 1 | Clash of the Titans |
| 2 | Forgetting Sarah Marshal |
| 2 | Daddy's Little Girls |
| 3 | Clash of the Titans |

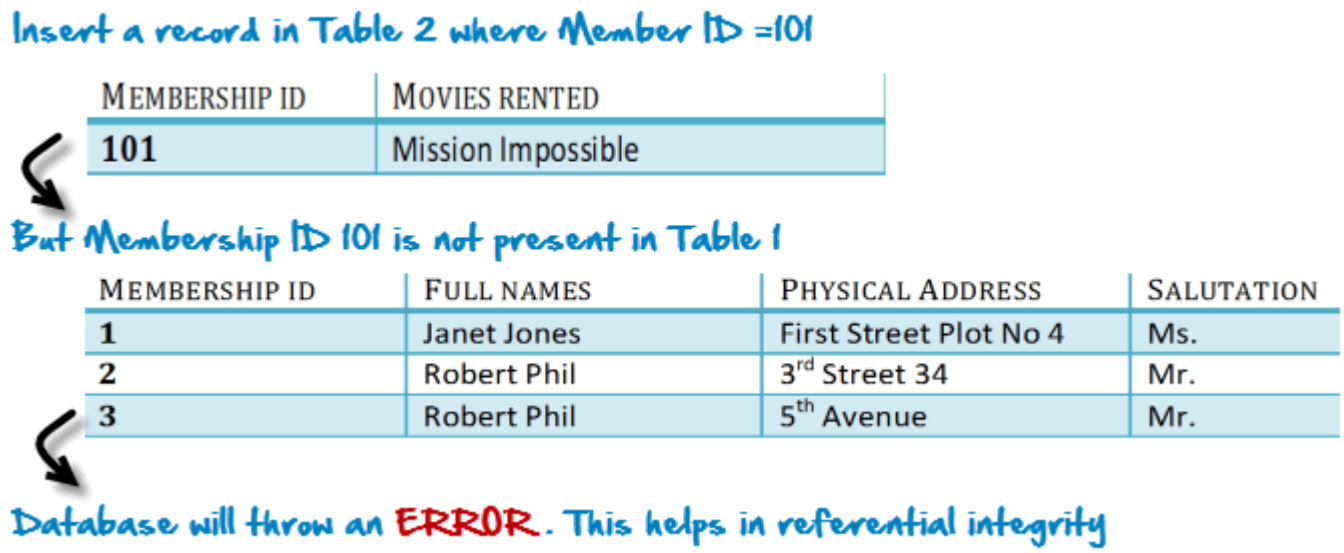
Foreign Key references the primary key of another Table! It helps connect your Tables

- A foreign key can have a different name from its primary key
- It ensures rows in one table have corresponding rows in another
- Unlike the Primary key, they do not have to be unique. Most often they aren't
- Foreign keys can be null even though primary keys can not




***Why do you need a foreign key?***

Suppose, a novice inserts a record in Table B such as




You will only be able to insert values into your foreign key that exist in the unique key in the parent table. This helps in referential integrity.

The above problem can be overcome by declaring membership id from Table2 as foreign key of membership id from Table1

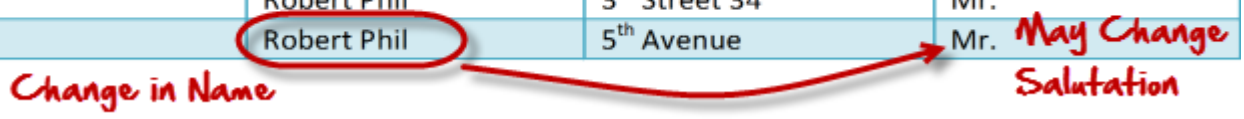
Now, if somebody tries to insert a value in the membership id field that does not exist in the parent table, an error will be shown!

**What are transitive functional dependencies?**

A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change

Consider the table 1. Changing the non-key column Full Name may change Salutation.

| MEMBERSHIP ID | FULL NAMES | PHYSICAL ADDRESS | SALUTATION |
|---|---|---|---|
| 1 | Janet Jones | First Street Plot No 4 | Ms. |
| 2 | Robert Phil | 3rd Street 34 | Mr. |
| 3 | Robert Phil | 5th Avenue | Mr. |




Let's move into 3NF

**3NF (Third Normal Form) Rules**

- Rule 1- Be in 2NF
- Rule 2- Has no transitive functional dependencies

To move our 2NF table into 3NF, we again need to again divide our table.

**3NF Example**

| MEMBERSHIP ID | FULL NAMES | PHYSICAL ADDRESS | SALUTATION ID |
|---|---|---|---|
| 1 | Janet Jones | First Street Plot No 4 | 2 |
| 2 | Robert Phil | 3rd Street 34 | 1 |
| 3 | Robert Phil | 5th Avenue | 1 |

| MEMBERSHIP ID | MOVIES RENTED |
|---|---|
| 1 | Pirates of the Caribbean |
| 1 | Clash of the Titans |
| 2 | Forgetting Sarah Marshal |
| 2 | Daddy's Little Girls |
| 3 | Clash of the Titans |

| SALUTATION ID | SALUTATION |
|---|---|
| 1 | Mr. |
| 2 | Ms. |
| 3 | Mrs. |
| 4 | Dr. |

We have again divided our tables and created a new table which stores Salutations.

There are no transitive functional dependencies, and hence our table is in 3NF

In Table 3 Salutation ID is primary key, and in Table 1 Salutation ID is foreign to primary key in Table 3

Now our little example is at a level that cannot further be decomposed to attain higher forms of normalization. In fact, it is already in higher normalization forms. Separate efforts for moving into next levels of normalizing data are normally needed in complex databases. However, we will be discussing next levels of normalizations in brief in the following.

**BCNF (Boyce-Codd Normal Form)**

Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one **Candidate** Key.

Sometimes is BCNF is also referred as **3.5 Normal Form.**

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

If no database table instance contains two or more, independent and multivalued data describing the relevant entity, then it is in 4th Normal Form.

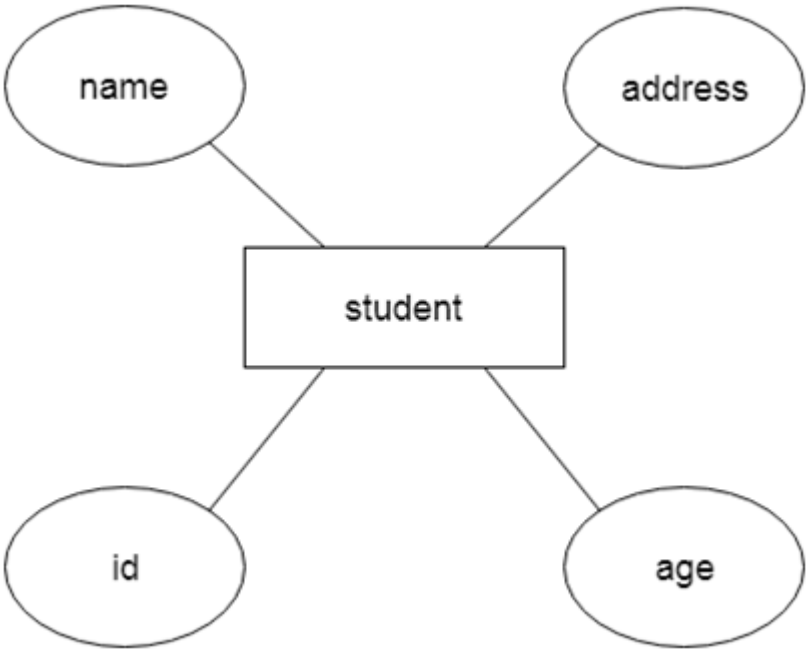
**5NF (Fifth Normal Form) Rules**

A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.

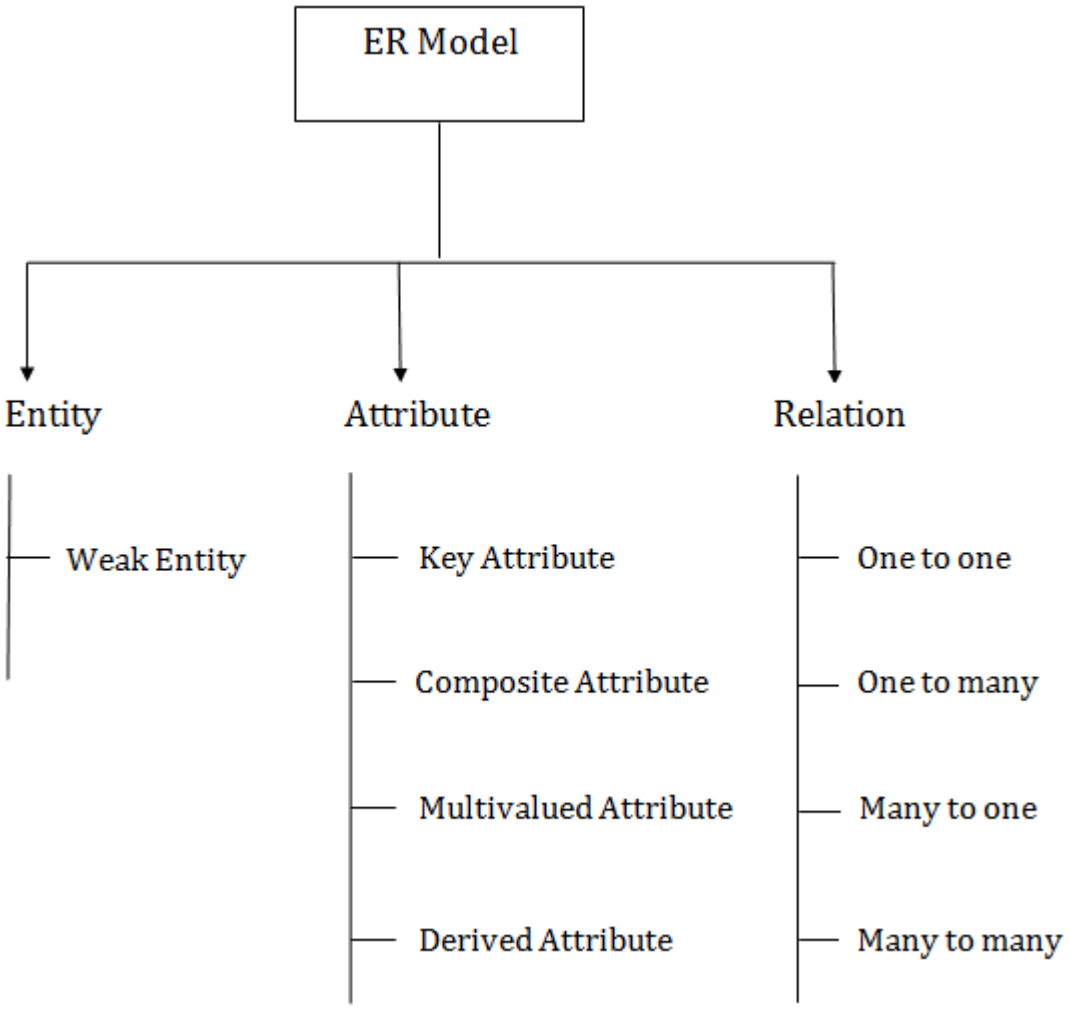
## 9.ER model

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

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


name
address
student
id
age

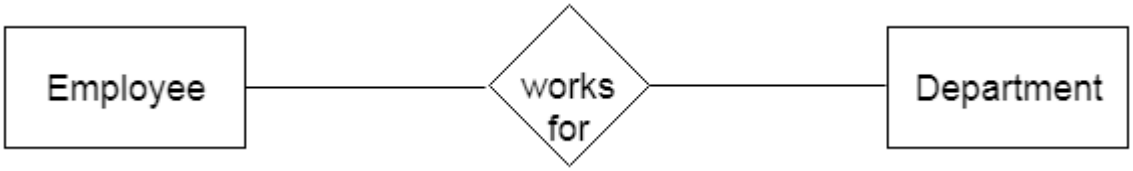
## Component of ER Diagram




### 1. Entity:

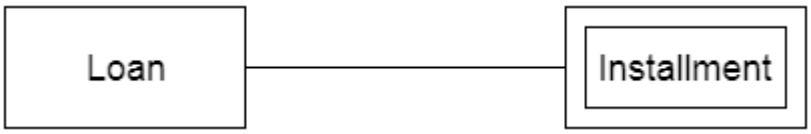
An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

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




**a. Weak Entity**

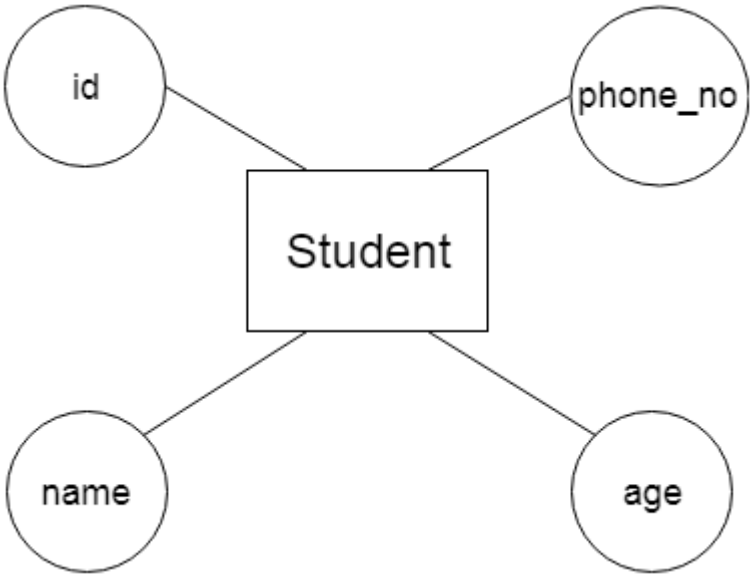
An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.




## 2. Attribute

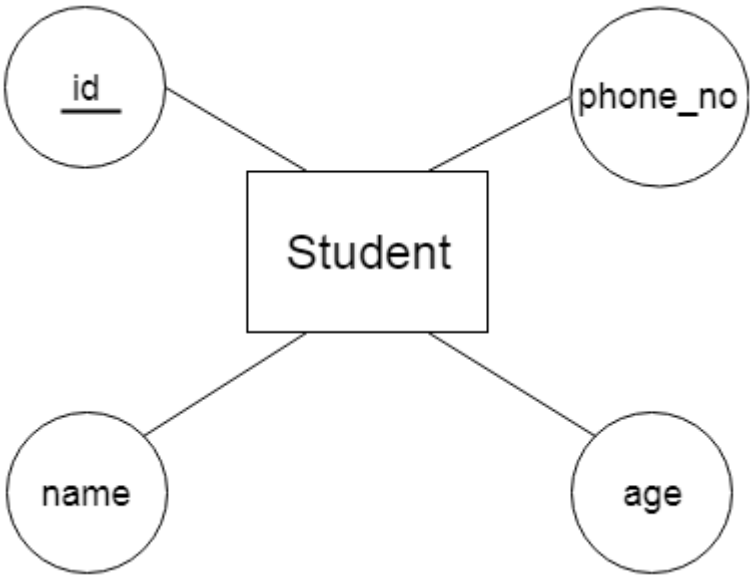
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

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



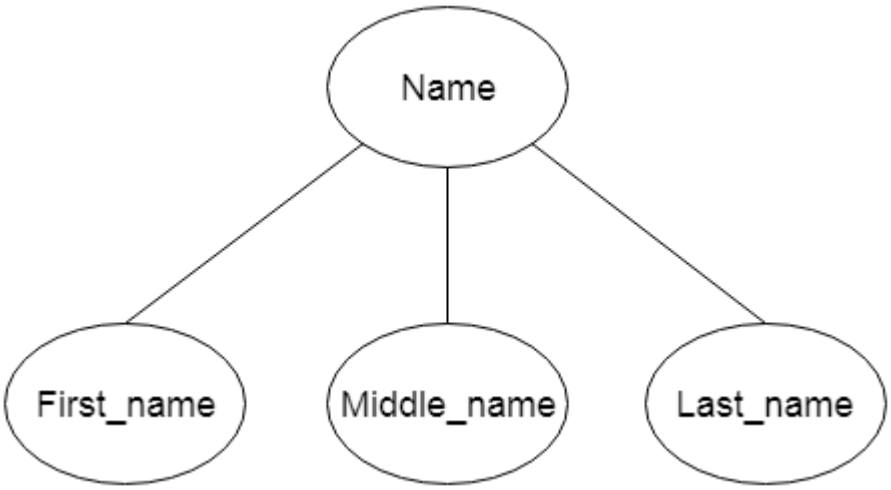

### a. Key Attribute

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



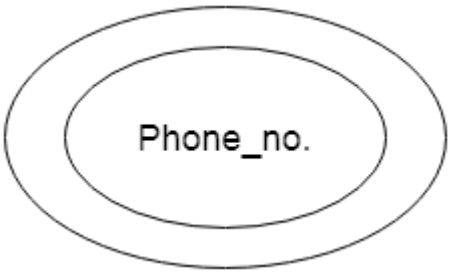

**b. Composite Attribute**

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




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

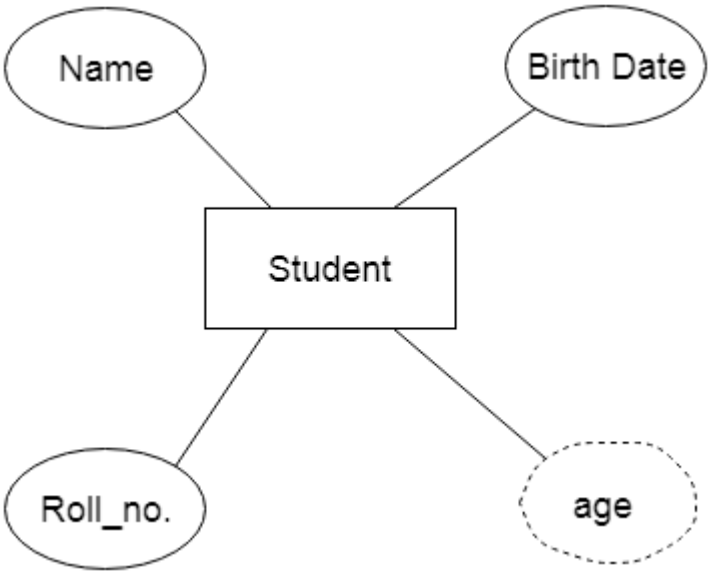
**For example,** a student can have more than one phone number.




**d. Derived Attribute**

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

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




## 3. Relationship

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

Types of relationship are as follows:

**a. One-to-One Relationship**

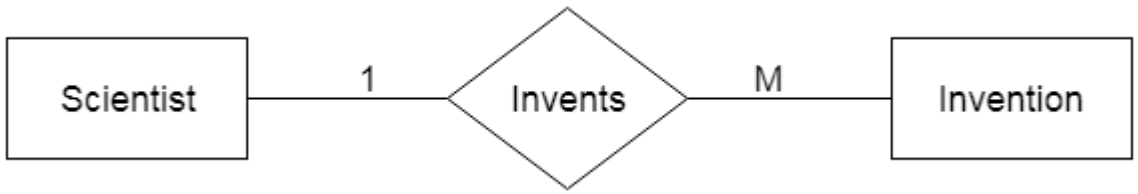
When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

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


**b. One-to-many relationship**

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

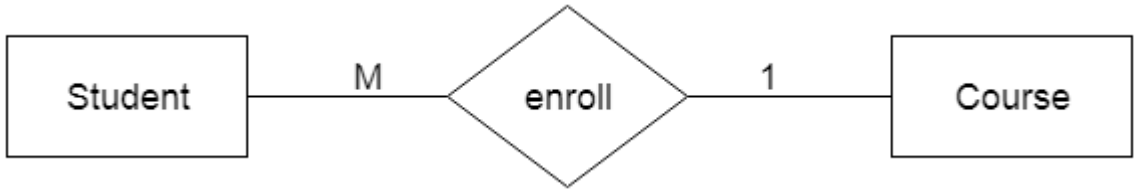
**For example,** Scientist can invent many inventions, but the invention is done by the only specific scientist.




**c. Many-to-one relationship**

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

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



**d. Many-to-many relationship**

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


## Notation of ER diagram

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

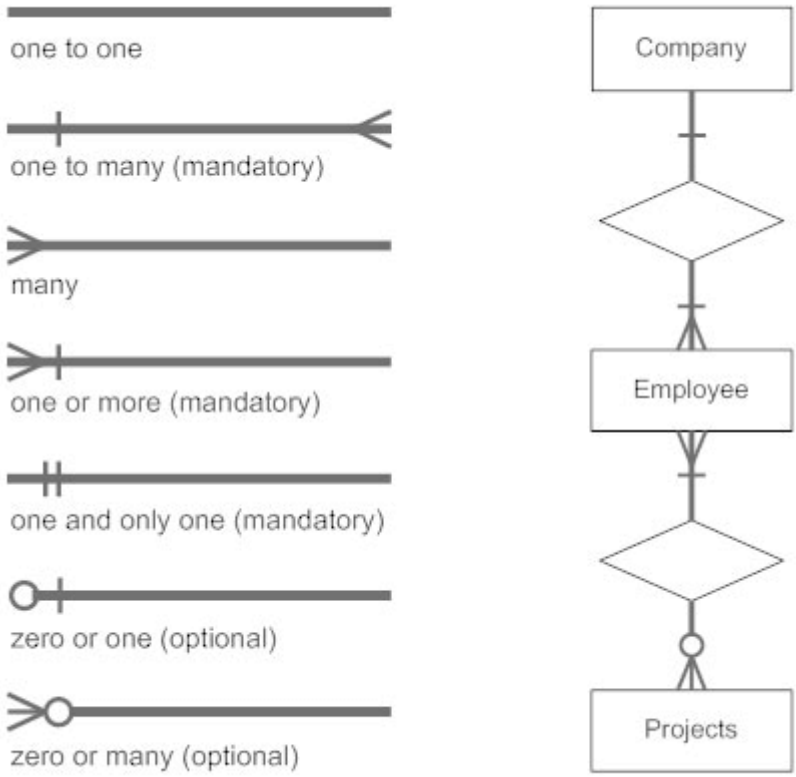



**Fig: Notations of ER diagram**

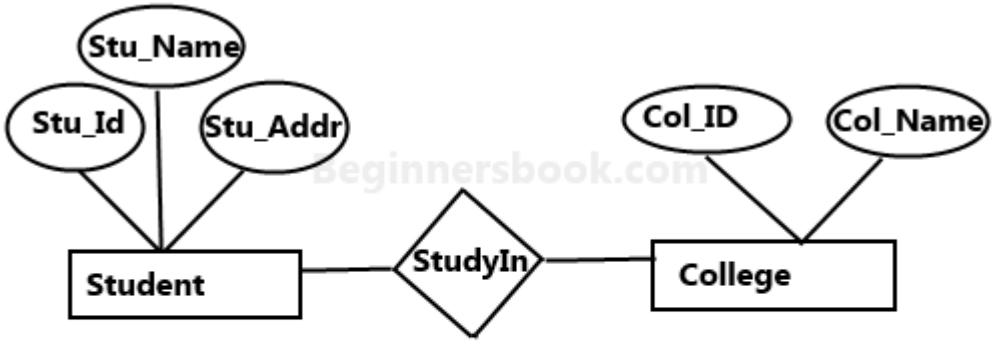
**10. Entity Relationship Diagram** – ER Diagram in DBMS: An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

**What is an Entity Relationship Diagram (ER Diagram)?**

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Lets have a look at a simple ER diagram to understand this concept.

**A simple ER Diagram:**

In the following diagram we have two entities Student and College and their relationship. The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time. Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.



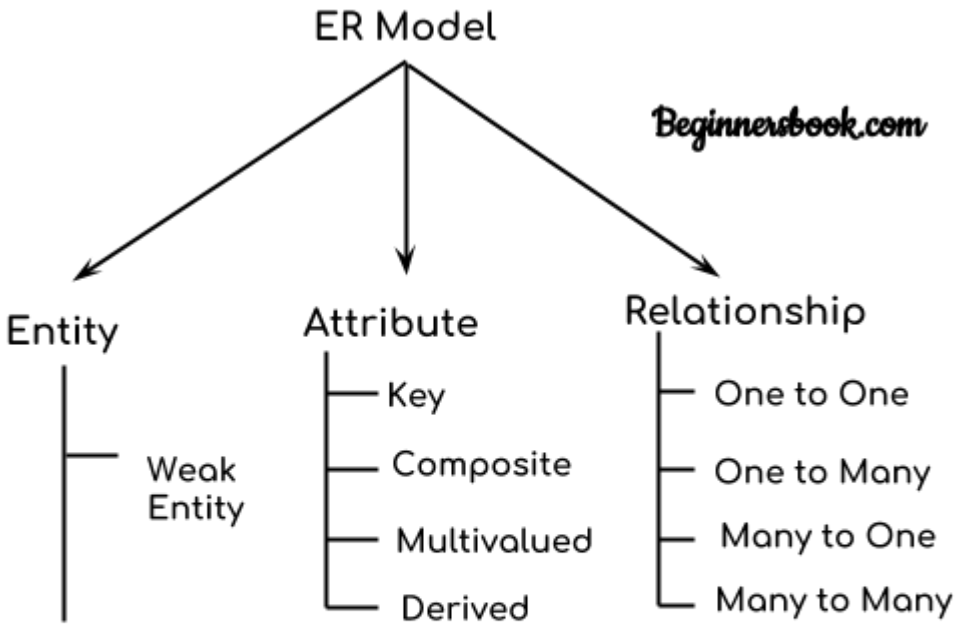

**Sample E-R Diagram**

Here are the geometric shapes and their meaning in an E-R Diagram. We will discuss these terms in detail in the next section(Components of a ER Diagram) of this guide so don't worry too much about these terms now, just go through them once.

**Rectangle**: Represents Entity sets.
**Ellipses**: Attributes
**Diamonds**: Relationship Set
**Lines**: They link attributes to Entity Sets and Entity sets to Relationship Set
**Double Ellipses:** Multivalued Attributes
**Dashed Ellipses**: Derived Attributes

**Double Rectangles**: Weak Entity Sets
**Double Lines**: Total participation of an entity in a relationship set

**Components of a ER Diagram**

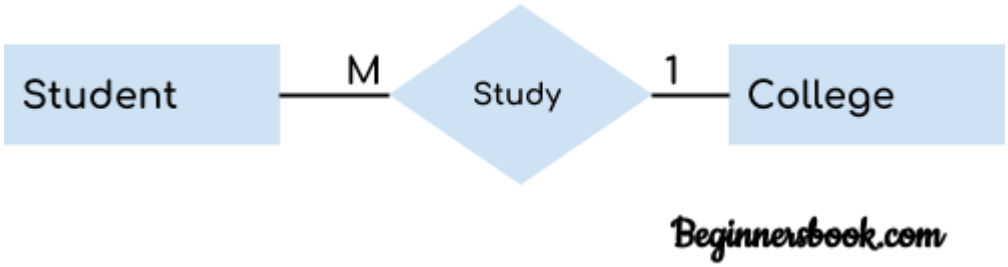



Components of ER Diagram

As shown in the above diagram, an ER diagram has three main components:
1. Entity
2. Attribute
3. Relationship

**1. Entity**

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.
For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college. We will read more about relationships later, for now focus on entities.




**Weak Entity:**
An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.


**2. Attribute**

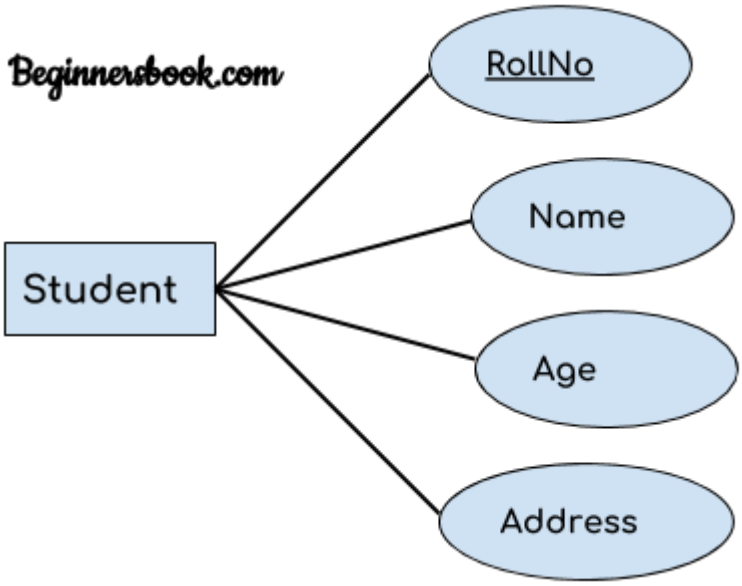
An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

***1. Key attribute:***

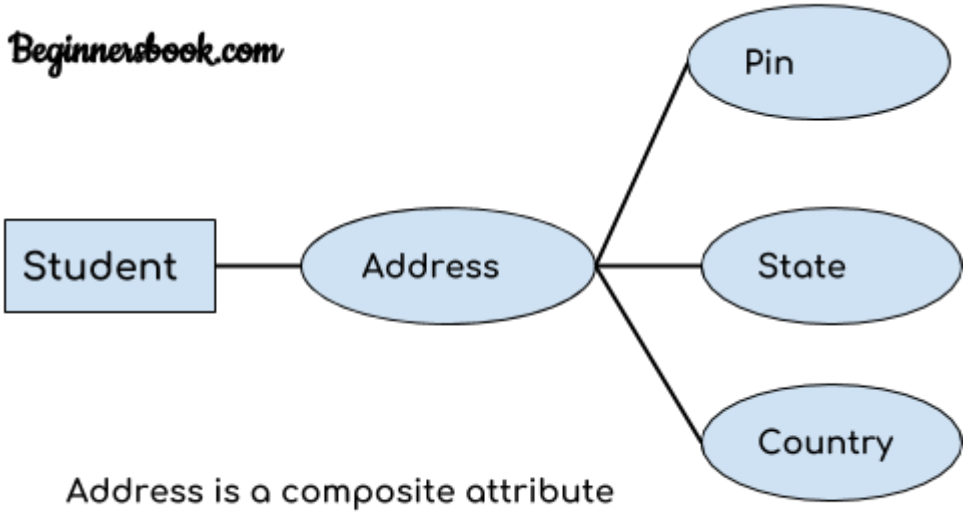
A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by

oval same as other attributes however the **text of key attribute is underlined**.




***2. Composite attribute:***

An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.



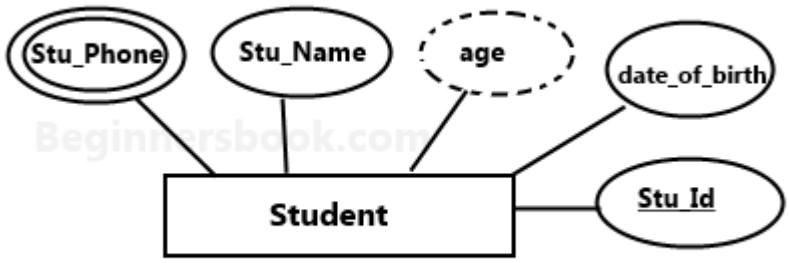

***3. Multivalued attribute:***

An attribute that can hold multiple values is known as multivalued attribute. It is represented with **double ovals** in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

*4. Derived attribute:*

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by **dashed oval** in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

**E-R diagram with multivalued and derived attributes**:

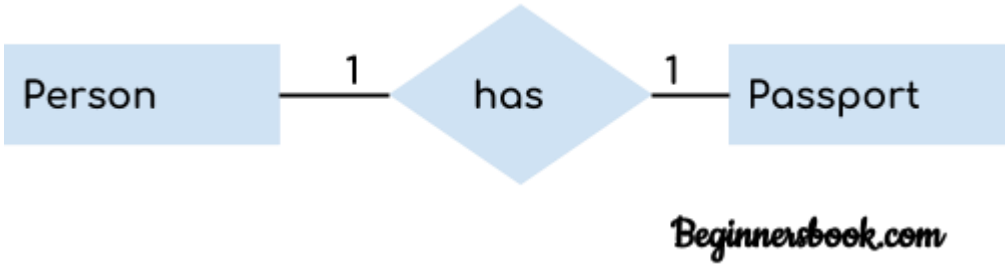



**3. Relationship**

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:
1. One to One
2. One to Many
3. Many to One
4. Many to Many

*1. One to One Relationship*

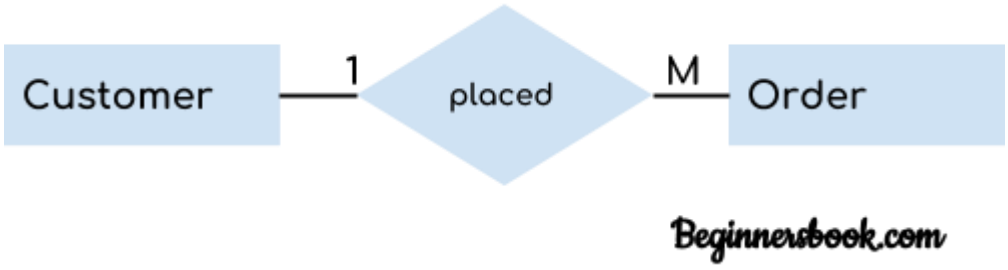
When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.




*2. One to Many Relationship*

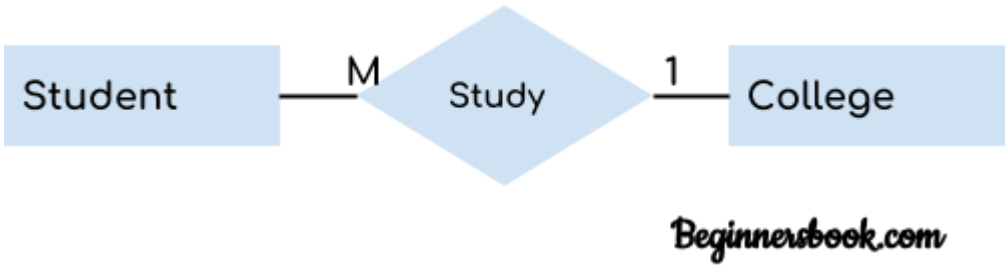
When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many

orders but a order cannot be placed by many customers.



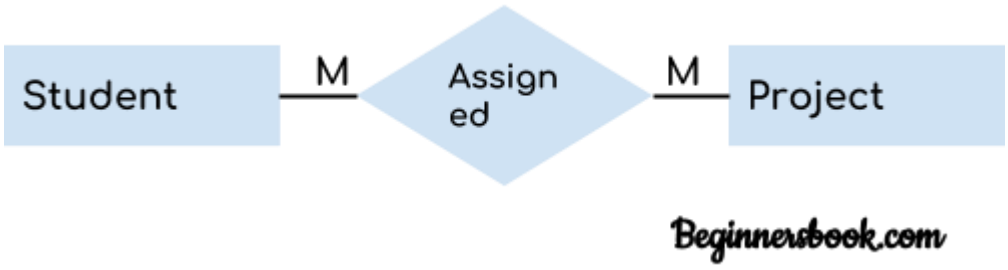

***3. Many to One Relationship***

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.




***4. Many to Many Relationship***

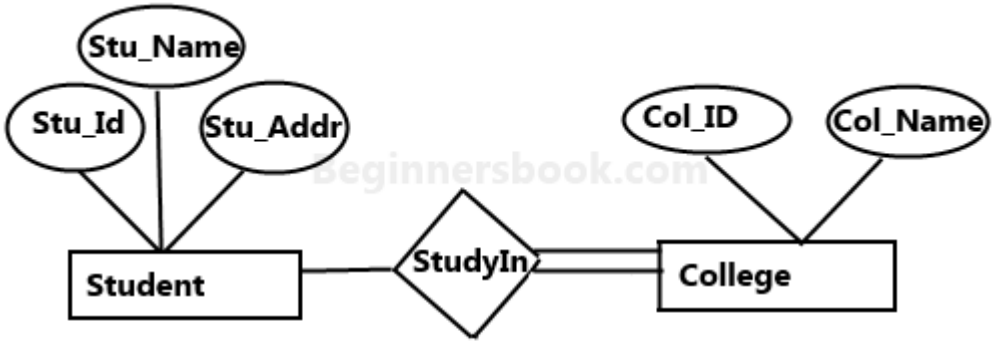
When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.




**Total Participation of an Entity set**

A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set. For example: In the below diagram each college must

have at-least one associated Student.



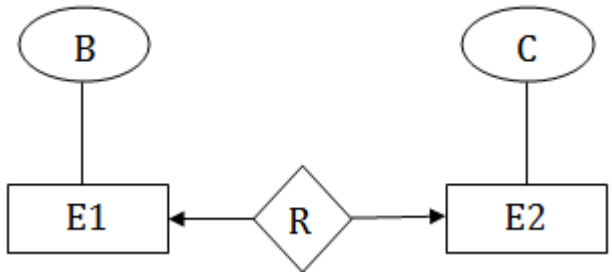

**E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.**

## 11.Constraints

- A mapping constraint is a data constraint that expresses the number of entities to which another entity can be related via a relationship set.
- It is most useful in describing the relationship sets that involve more than two entity sets.
- For binary relationship set R on an entity set A and B, there are four possible mapping cardinalities. These are as follows:
    1. One to one (1:1)
    2. One to many (1:M)
    3. Many to one (M:1)
    4. Many to many (M:M)

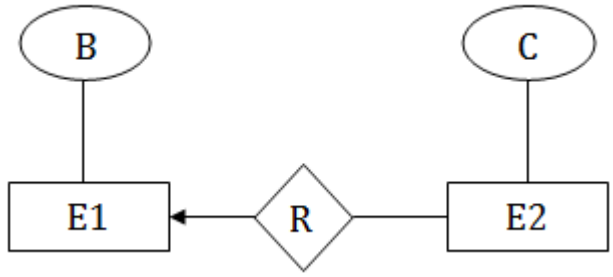
### One-to-one

In one-to-one mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with at most one entity in E1.



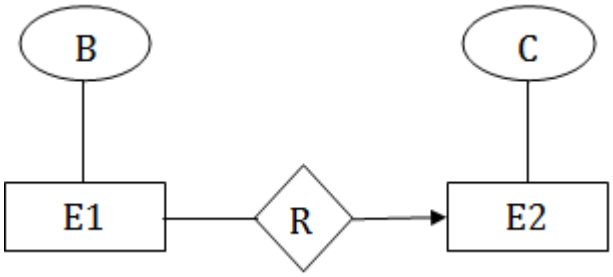

## One-to-many

In one-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with at most one entity in E1.



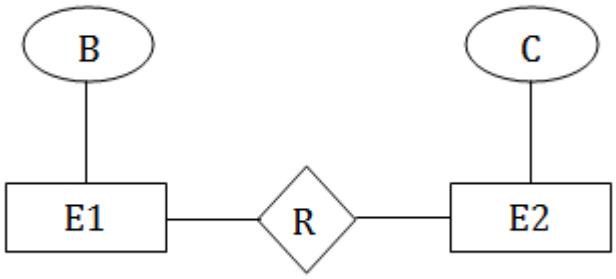

## Many-to-one

In one-to-many mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with any number of entities in E1.




## Many-to-many

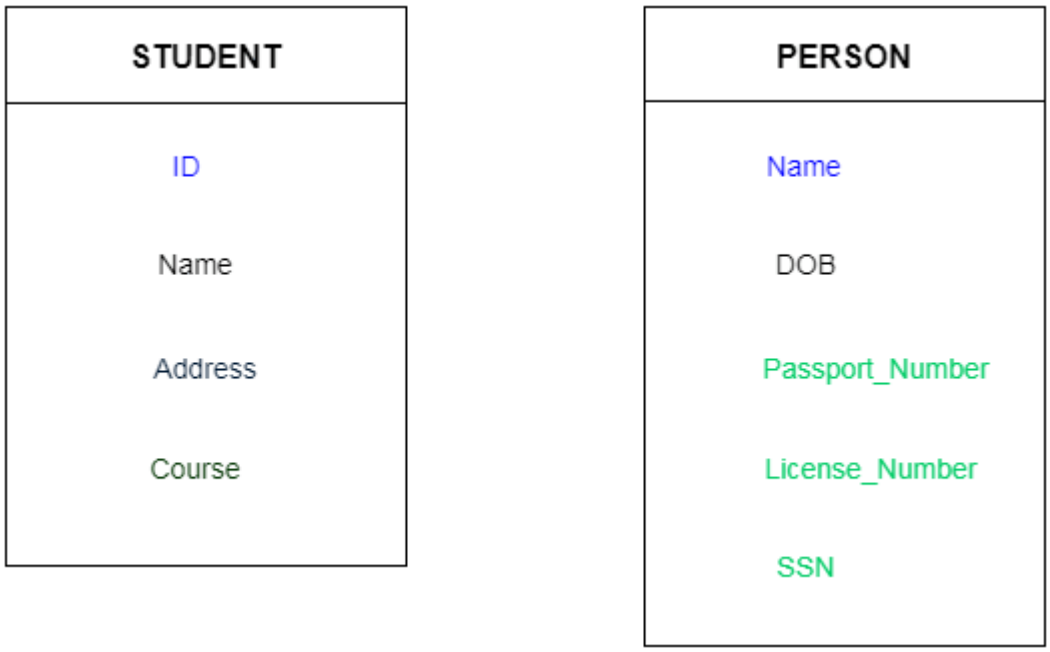
In many-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with any number of entities in E1.



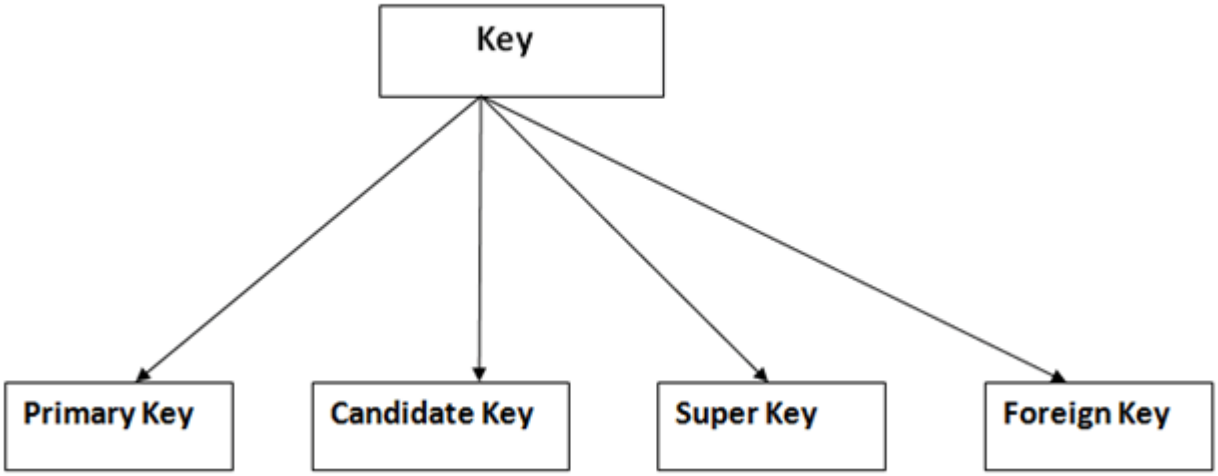

## Keys

- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

**For example:** In Student table, ID is used as a key because it is unique for each student. In PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.

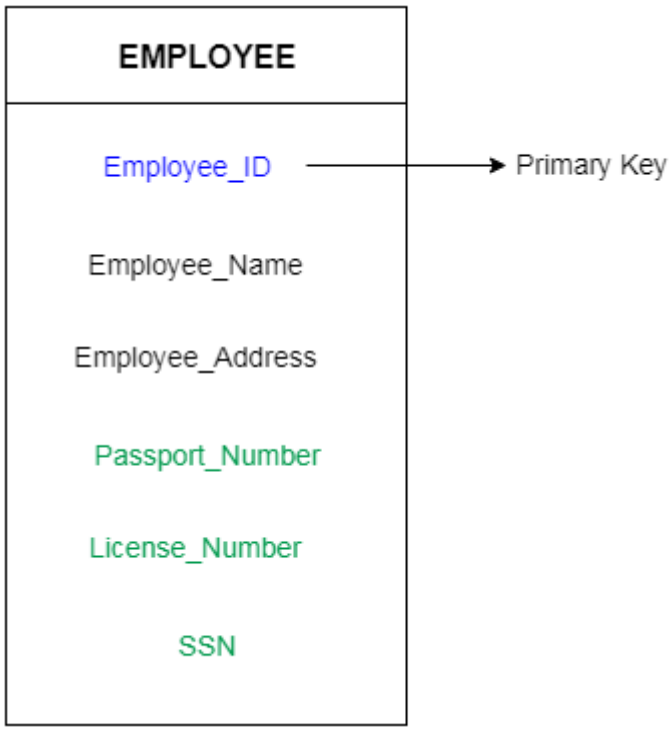


Types of key:




1. Primary key

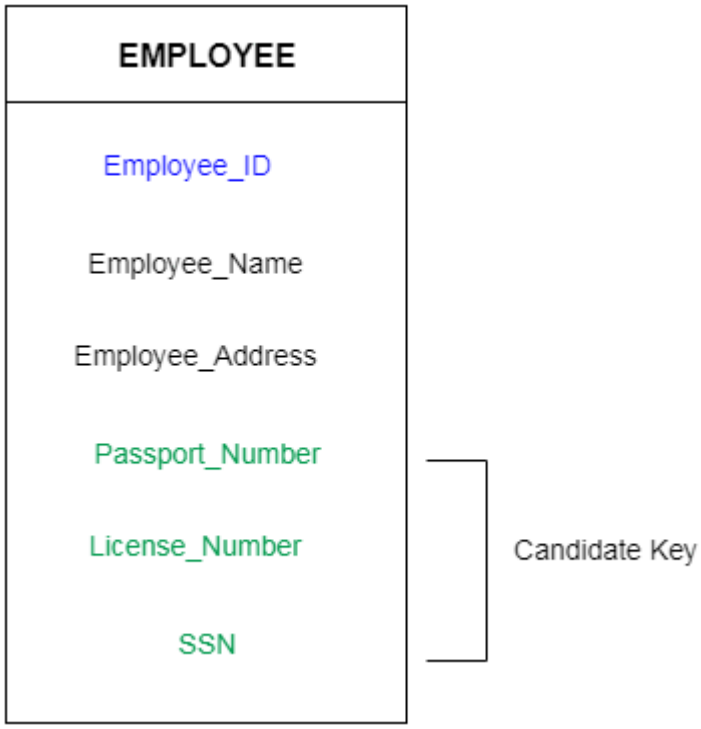
- It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys as we saw in PERSON table. The key which is most suitable from those lists become a primary key.
- In the EMPLOYEE table, ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary key since they are also unique.
- For each entity, selection of the primary key is based on requirement and developers.




## 2. Candidate key

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

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




### 3. Super Key

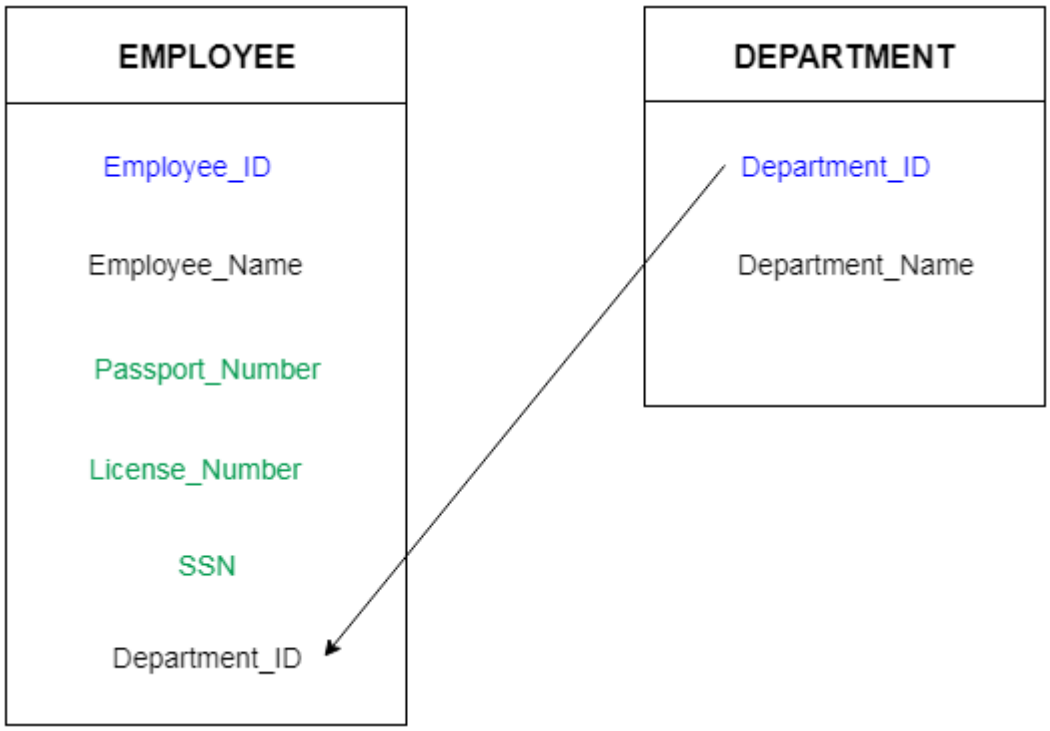
Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.

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

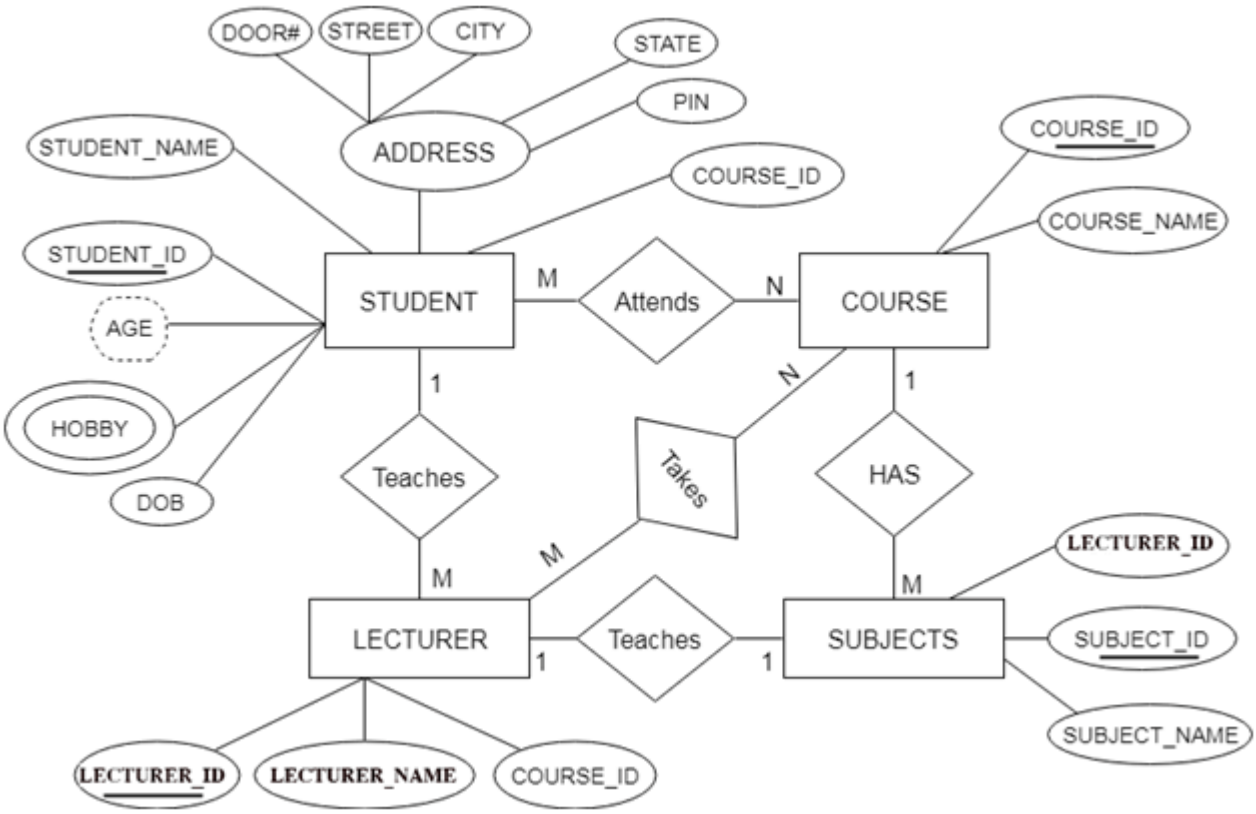
The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

### 4. Foreign key

- Foreign keys are the column of the table which is used to point to the primary key of another table.
- In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.




**The 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.

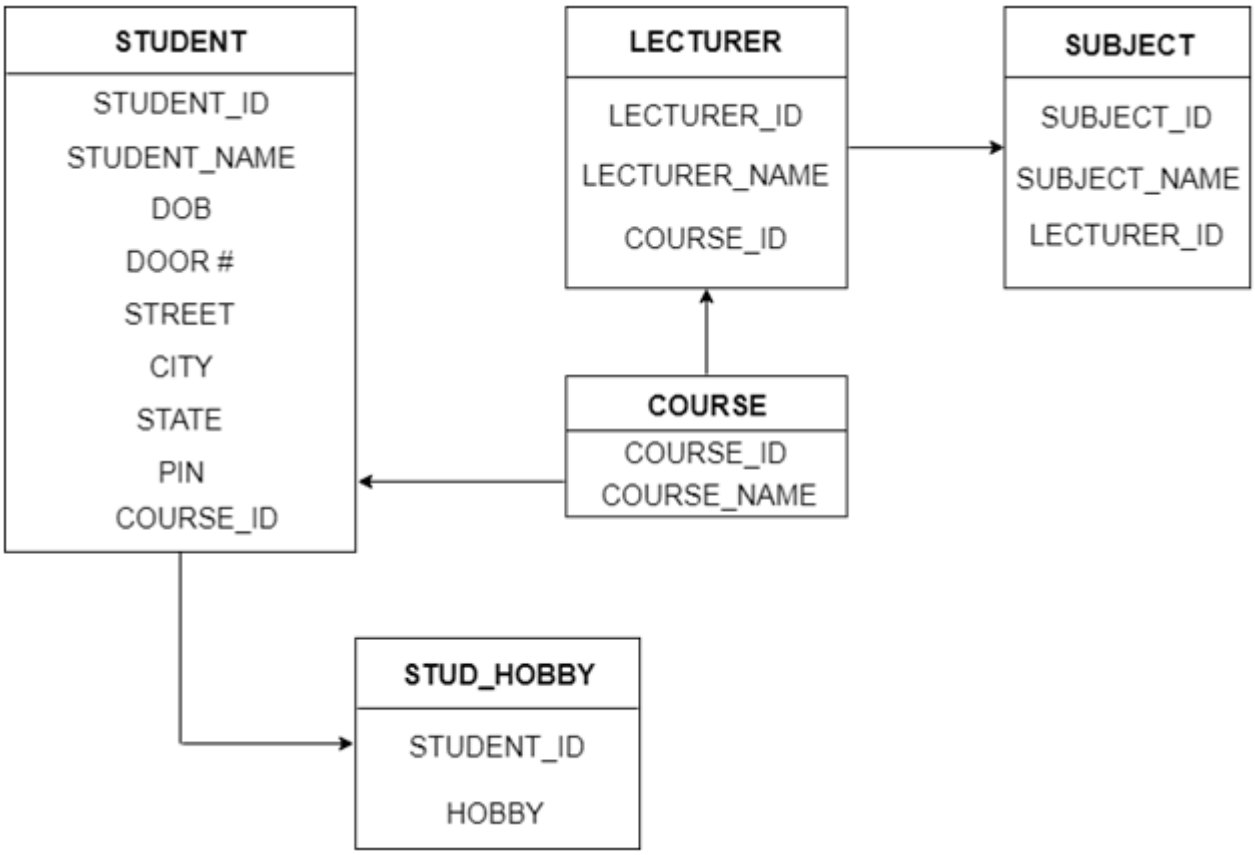
- **Composite attribute represented by components.**

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:




## 12.ER Design Issues

In the previous sections of the data modeling, we learned to design an ER diagram. We also discussed different ways of defining entity sets and relationships among them. We also understood the various designing shapes that represent a relationship, an entity, and its attributes. However, users often mislead the concept of the elements and the design process of the ER diagram. Thus, it leads to a complex structure of the ER diagram and certain issues that does not meet the characteristics of the real-world enterprise model.

Here, we will discuss the basic design issues of an ER database schema in the following points:

### 1) Use of Entity Set vs Attributes

The use of an entity set or attribute depends on the structure of the real-world enterprise that is being modelled and the semantics associated with its attributes. It leads to a mistake when the user use the primary key of an entity set as an attribute of another entity set. Instead, he should use the relationship to do so. Also, the primary key attributes are implicit in the relationship set, but we designate it in the relationship sets.

### 2) Use of Entity Set vs. Relationship Sets

It is difficult to examine if an object can be best expressed by an entity set or relationship set. To understand and determine the right use, the user need to designate a relationship set for describing an action that occurs in-between the entities. If there is a requirement of representing the object as a relationship set, then its better not to mix it with the entity set.

### 3) Use of Binary vs n-ary Relationship Sets

Generally, the relationships described in the databases are binary relationships. However, non-binary relationships can be represented by several binary relationships. For example, we can create and represent a ternary relationship 'parent' that may relate to a child, his father, as well as his mother. Such relationship can also be represented by two binary relationships i.e, mother and father, that may relate to their child. Thus, it is possible to represent a non-binary relationship by a set of distinct binary relationships.

### 4) Placing Relationship Attributes

The cardinality ratios can become an affective measure in the placement of the relationship attributes. So, it is better to associate the attributes of one-to-one or one-to-many relationship sets with any participating entity sets, instead of any relationship set. The decision of placing the specified attribute as a relationship or entity attribute should possess the charactestics of the real world enterprise that is being modelled.

**For example**, if there is an entity which can be determined by the combination of participating entity sets, instead of determing it as a separate entity. Such type of attribute must be associated with the many-to-many relationship sets.

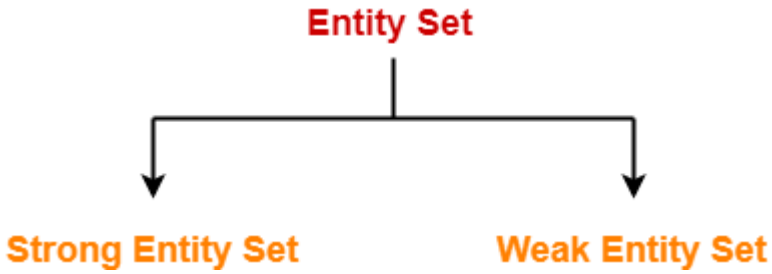
Thus, it requires the overall knowledge of each part that is involved inb desgining and modelling an ER diagram. The basic requirement is to analyse the real-world enterprise and the connectivity of one entity or attribute with other.

Entity Set in DBMS- An entity refers to any object having-

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

**Types of Entity Sets-**

An entity set may be of the following two types-




1. Strong entity set
2. Weak entity set

**1. Strong Entity Set-**

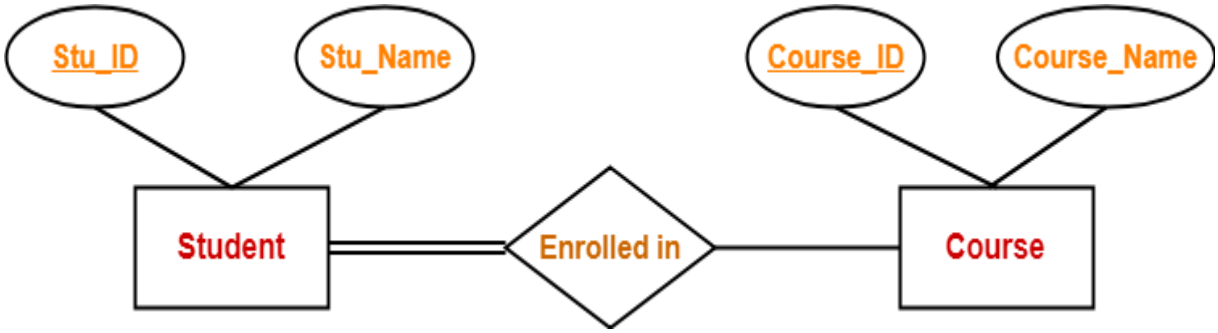
- A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities.
- In other words, a primary key exists for a strong entity set.
- Primary key of a strong entity set is represented by underlining it.

**Symbols Used-**

- A single rectangle is used for representing a strong entity set.
- A diamond symbol is used for representing the relationship that exists between two strong entity sets.
- A single line is used for representing the connection of the strong entity set with the relationship set.
- A double line is used for representing the total participation of an entity set with the relationship set.
- Total participation may or may not exist in the relationship.

**Example-**

Consider the following ER diagram-




In this ER diagram,

- Two strong entity sets "**Student**" and "**Course**" are related to each other.
- Student ID and Student name are the attributes of entity set "Student".
- Student ID is the primary key using which any student can be identified uniquely.
- Course ID and Course name are the attributes of entity set "Course".
- Course ID is the primary key using which any course can be identified uniquely.
- Double line between Student and relationship set signifies total participation.
- It suggests that each student must be enrolled in at least one course.
- Single line between Course and relationship set signifies partial participation.
- It suggests that there might exist some courses for which no enrollments are made.

## 2. Weak Entity Set-

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

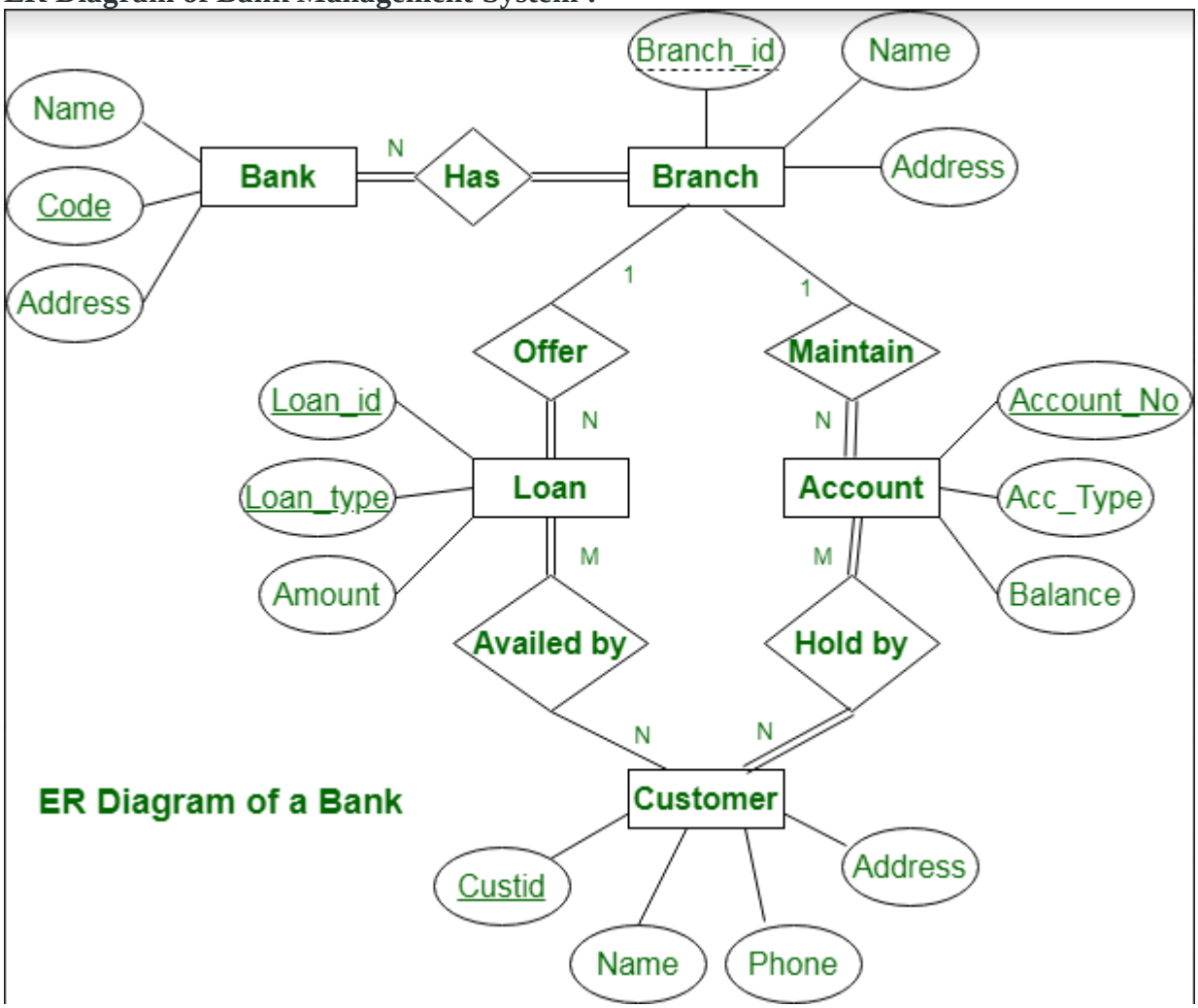
**ER diagram of Bank Management System**

ER diagram is known as Entity-Relationship diagram. It is used to analyze to structure of the Database. It shows relationships between entities and their attributes. An ER model provides a means of communication.
ER diagram of Bank has the following description :

- Bank have Customer.
- Banks are identified by a name, code, address of main office.
- Banks have branches.
- Branches are identified by a branch_no., branch_name, address.
- Customers are identified by name, cust-id, phone number, address.
- Customer can have one or more accounts.
- Accounts are identified by acc_no., acc_type, balance.

- Customer can avail loans.
- Loans are identified by loan_id, loan_type and amount.
- Account and loans are related to bank's branch.

**ER Diagram of Bank Management System :**




This bank ER diagram illustrates key information about bank, including entities such as branches, customers, accounts, and loans. It allows us to understand the relationships between entities.

**Entities** and their **Attributes** are :

- **Bank Entity :** Attributes of Bank Entity are Bank Name, Code and Address.
  Code is Primary Key for Bank Entity.
- **Customer Entity :** Attributes of Customer Entity are Customer_id, Name, Phone Number and Address.
  Customer_id is Primary Key for Customer Entity.
- **Branch Entity :** Attributes of Branch Entity are Branch_id, Name and Address.
  Branch_id is Primary Key for Branch Entity.
- **Account Entity :** Attributes of Account Entity are Account_number, Account_Type and Balance.
  Account_number is Primary Key for Account Entity.
- **Loan Entity :** Attributes of Loan Entity are Loan_id, Loan_Type and Amount.
  Loan_id is Primary Key for Loan Entity.

Relationships are :

- **Bank has Branches => 1 : N**
  One Bank can have many Branches but one Branch can not belong to many Banks, so the relationship between Bank and Branch is one to many relationship.
- **Branch maintain Accounts => 1 : N**
  One Branch can have many Accounts but one Account can not belong to many Branches, so the relationship between Branch and Account is one to many relationship.
- **Branch offer Loans => 1 : N**
  One Branch can have many Loans but one Loan can not belong to many Branches, so the relationship between Branch and Loan is one to many relationship.
- **Account held by Customers => M : N**
  One Customer can have more than one Accounts and also One Account can be held by one or more Customers, so the relationship between Account and Customers is many to many relationship.
- **Loan availed by Customer => M : N**
  (Assume loan can be jointly held by many Customers).
  One Customer can have more than one Loans and also One Loan can be availed by one or more Customers, so the relationship between Loan and Customers is many to many relationship.

**12.Unified Modeling Language (UML)** is a general purpose modelling language. The main aim of UML is to define a standard way to **visualize** the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.
UML is **not a programming language**, it is rather a visual language. We use UML diagrams to portray the **behavior and structure** of a system. UML helps software engineers, businessmen and system architects with modelling, design and analysis. The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997. Its been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005. UML has been revised over the years and is reviewed periodically.

**Do we really need UML?**

- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- Businessmen do not understand code. So UML becomes essential to communicate with non programmers essential requirements, functionalities and processes of the system.
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.

UML is linked with **object oriented** design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

1. **Structural Diagrams –** Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
2. **Behavior Diagrams –** Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

## 13.Structure of Relational Databases

A relational database consists of a collection of **tables**, each of which is assigned a unique name.

| *account-number* | *branch-name* | *balance* |
|---|---|---|
| A-101 | Downtown | 500 |
| A-102 | Perryridge | 400 |
| A-201 | Brighton | 900 |
| A-215 | Mianus | 700 |
| A-217 | Brighton | 750 |
| A-222 | Redwood | 700 |
| A-305 | Round Hill | 350 |

**Figure 3.1** The *account* relation.

A row in a table represents a relationship among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of table and the mathematical concept of relation, from which the relational data model takes its name. In what follows, we introduce the concept of relation.

**Basic Structure**

Consider the account table of Figure 3.1. It has three column headers: account-number, branch-name, and balance. Following the terminology of the relational model, these headers are **attributes**. For each attribute, there is a set of permitted values, called the **domain** of that attribute. For the attribute branch-name, for example, the domain is the set of all branch names.

LetD1 denote the set of all account numbers, D2 the set of all branch names, and D3the set of all balances. Any row of account must consist of a 3-tuple (v1, v2, v3), where v1 is an account number (that is, v1 is in domain D1),v2 is a branch name (that is, v2 is in domain D2), and v3 is a balance (that is, v3 is in domain D3). In general, account will contain only a subset of the set of all possible rows. Therefore, account is a subset of

| *account-number* | *branch-name* | *balance* |
|---|---|---|
| A-101 | Downtown | 500 |
| A-215 | Mianus | 700 |
| A-102 | Perryridge | 400 |
| A-305 | Round Hill | 350 |
| A-201 | Brighton | 900 |
| A-222 | Redwood | 700 |
| A-217 | Brighton | 750 |

**Figure 3.2** The *account* relation with unordered tuples.

$$D_1 \times D_2 \times D_3$$

In general, a **table** of *n* attributes must be a subset of

$$D_1 \times D_2 \times \cdots \times D_{n-1} \times D_n$$

Mathematicians define a **relation** to be a subset of a Cartesian product of a list of domains. This definition corresponds almost exactly with our definition of table. The only difference is that we have assigned names to attributes, whereas mathematicians rely on numeric "names," using the integer 1 to denote the attribute whose domain appears first in the list of domains, 2 for the attribute whose domain appears second, and so on. Because tables are essentially relations, we shall use the mathematical terms relation and tuple in place of the terms table and row. A tuple variable is a variable that stands for a tuple; in other words, a tuple variable is a variable whose domain is the set of all tuples.

In the account relation of Figure 3.1, there are seven tuples. Let the tuple variable t refer to the first tuple of the relation. We use the notation t[account-number] to denote the value of t on the account-number attribute. Thus, t[account-number] = "A-101," and t[branch-name] = "Downtown". Alternatively, we may write t[1] to denote the value of tuple t on the first attribute (account-number), t[2] to denote branch-name, and so on. Since a relation is a set of tuples, we use the mathematical notation of t ∈r to denote that tuple t is in relation r.

The order in which tuples appear in a relation is irrelevant, since a relation is a set of tuples. Thus, whether the tuples of a relation are listed in sorted order, as in Figure 3.1, or are unsorted, as in Figure 3.2, does not matter; the relations in the two figures above are the same, since both contain the same set of tuples.

For all relations r, the domains of all attributes of r be atomic. A domain is atomic if elements of the domain are considered to be indivisible units. For example, the set of integers is an atomic domain, but the set of all sets of integers is a non atomic domain.

**Database Schema**

The concept of a relation corresponds to the programming-language notion of a variable. The concept of a relation schema Corresponds to the programming-language notion of type definition.

It is convenient to give a name to a relation schema, just as we give names to type definitions in programming languages. We adopt the convention of using lowercase names for relations, and names beginning with an uppercase letter for relation schemas. Following this notation, we use Account-schema to denote the relation schema for relation account. Thus,

| *branch-name* | *branch-city* | *assets* |
|---|---|---|
| Brighton | Brooklyn | 7100000 |
| Downtown | Brooklyn | 9000000 |
| Mianus | Horseneck | 400000 |
| North Town | Rye | 3700000 |
| Perryridge | Horseneck | 1700000 |
| Pownal | Bennington | 300000 |
| Redwood | Palo Alto | 2100000 |
| Round Hill | Horseneck | 8000000 |

**Figure 3.3** The *branch* relation.

$$Account\text{-}schema = (account\text{-}number, branch\text{-}name, balance)$$

The fact that account is a relation on Account-schema by

$$account(Account\text{-}schema)$$

The concept of a relation instance corresponds to the programming language notion of a value of a variable. The value of a given variable may change with time; similarly, the contents of a relation instance may change with time as the relation is updated. However, we often simply say "relation" when we actually mean "relation instance."

The schema for that relation is

$$Branch\text{-}schema = (branch\text{-}name, branch\text{-}city, assets)$$

Note that the attribute branch-name appears in both Branch-schema and Account schema. This duplication is not a coincidence. Rather, using common attributes in relation schemas is one way of relating tuples of distinct relations. For example, suppose we wish to find the information about all of the accounts maintained in branches located in Brooklyn. We look first at the branch relation to find the names of all the branches located in Brooklyn. Then, for each such branch, we would look in the account relation to find the information about the accounts maintained at that branch.

This is not surprising recall that the primary key attributes of a strong entity set appear in the table created to represent the entity set, as well as in the tables created to represent relationships that the entity set participates in.

| customer-name | customer-street | customer-city |
|---|---|---|
| Adams | Spring | Pittsfield |
| Brooks | Senator | Brooklyn |
| Curry | North | Rye |
| Glenn | Sand Hill | Woodside |
| Green | Walnut | Stamford |
| Hayes | Main | Harrison |
| Johnson | Alma | Palo Alto |
| Jones | Main | Harrison |
| Lindsay | Park | Pittsfield |
| Smith | North | Rye |
| Turner | Putnam | Stamford |
| Williams | Nassau | Princeton |

**Figure 3.4** The *customer* relation.

Let us continue our banking example. We need a relation to describe information about customers. The relation schema is

$$Customer\text{-}schema = (customer\text{-}name, customer\text{-}street, customer\text{-}city)$$

Figure 3.4 shows a sample relation customer (Customer-schema).We assume that the customer name uniquely identifies a customer obviously this may not be true in the real world, but the assumption makes our examples much easier to read.

In a real-world database, the customer-id (which could be a social-security number, or an identifier generated by the bank) would serve to uniquely identify customers. We need a relation to describe the association between customers and accounts.

The relation schema to describe this association is

| customer-name | account-number |
|---|---|
| Hayes | A-102 |
| Johnson | A-101 |
| Johnson | A-201 |
| Jones | A-217 |
| Lindsay | A-222 |
| Smith | A-215 |
| Turner | A-305 |

**Figure 3.5** The *depositor* relation.

$$Depositor\text{-}schema = (customer\text{-}name, account\text{-}number)$$

Figure 3.5 shows a sample relation depositor (Depositor-schema). It would appear that, for our banking example, we could have just one relation schema, rather than several. That is, it may be easier for a user to think in terms of one relation schema, rather than in terms of several. Suppose that we used only one relation for our example, with schema

(*branch-name, branch-city, assets, customer-name, customer-street*
*customer-city, account-number, balance*)

Observe that, if a customer has several accounts, we must list her address once for each account. That is, we must repeat certain information several times. This repetition is wasteful and is avoided by the use of several relations, as in our example.

Schema Diagram A database schema, along with primary key and foreign key dependencies, can be depicted pictorially by schema diagrams. Figure 3.9 shows the schema diagram for our banking enterprise. Each relation appears as a box, with the attributes listed inside it and the relation name above it. If there are primary key attributes, a horizontal line crosses the box, with the primary key attributes listed above the line. Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation.

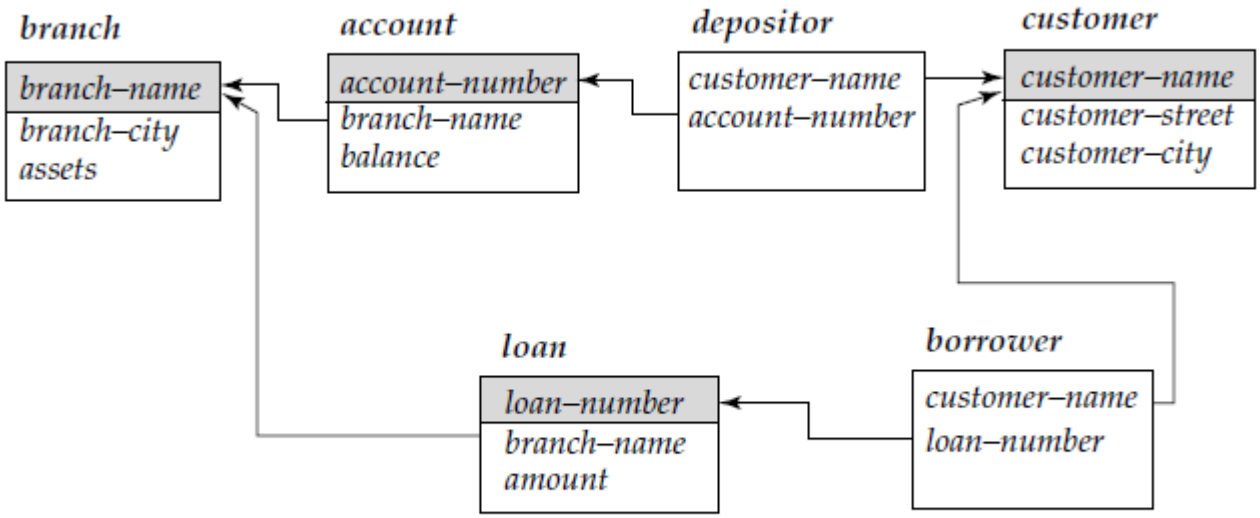



**Figure 3.9** Schema diagram for the banking enterprise.

Do not confuse a schema diagram with an E-R diagram. In particular, E-R diagrams do not show foreign key attributes explicitly, whereas schema diagrams show them explicity.

Many database systems provide design tools with a graphical user interface for creating schema diagrams.

## 14.The Relational Algebra

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result. The fundamental operations in the relational algebra are select, project, union, set difference, Cartesian product, and rename.

**Fundamental Operations :-**The select, project, and rename operations are called unary operations, because they operate on one relation. The other three operations operate on pairs of relations and are therefore, called binary operations.

| *loan-number* | *branch-name* | *amount* |
|---|---|---|
| L-11 | Round Hill | 900 |
| L-14 | Downtown | 1500 |
| L-15 | Perryridge | 1500 |
| L-16 | Perryridge | 1300 |
| L-17 | Downtown | 1000 |
| L-23 | Redwood | 2000 |
| L-93 | Mianus | 500 |

**Figure 3.6** The *loan* relation.

## The Select Operation

The select operation selects tuples that satisfy a given predicate. We use the lowercase Greek letter sigma (σ) to denote selection. The predicate appears as a subscript to σ. The argument relation is in parentheses after the σ. Thus, to select those tuples of the loan relation where the branch is "Perryridge," we write

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

If the loan relation is as shown in Figure 3.6, then the relation that results from the preceding query is as shown in Figure 3.10.

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

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

In general, we allow comparisons using =, ≠, <, ≤, >, ≥ in the selection predicate. Furthermore, we can combine several predicates into a larger predicate by using the connectives and (∧), or (∨), and not (¬). Thus, to find those tuples pertaining to loans of more than $1200 made by the Perryridge branch, we write

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

| *loan-number* | *branch-name* | *amount* |
|---|---|---|
| L-15 | Perryridge | 1500 |
| L-16 | Perryridge | 1300 |

**Figure 3.10** Result of $\sigma_{branch\text{-}name = \text{“Perryridge”}}(loan)$.

The selection predicate may include comparisons between two attributes. Consider the relation loan-officer that consists of three attributes: customer-name, banker-name, and loan-number, which specifies that a particular banker is the loan officer for a loan that belongs to some customer. To find all customers who have the same name as their loan officer, we can write

$$\sigma_{customer\text{-}name = banker\text{-}name}(loan\text{-}officer)$$

**Composition of Relational Operations**

---

The fact that the result of a relational operation is itself a relation is important. Consider the more complicated query "Find those customers who live in Harrison." We write:

$$\Pi_{customer\text{-}name}\ (\sigma_{customer\text{-}city = \text{“Harrison”}}\ (customer))$$

Notice that, instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

| *loan-number* | *amount* |
|---|---|
| L-11 | 900 |
| L-14 | 1500 |
| L-15 | 1500 |
| L-16 | 1300 |
| L-17 | 1000 |
| L-23 | 2000 |
| L-93 | 500 |

Loan number and the amount of the loan.

In general, since the result of a relational-algebra operation is of the same type (relation) as its inputs, relational-algebra operations can be composed together into a relational-algebra expression. Composing relational-algebra operations into relational- algebra expressions is just like composing arithmetic operations (such as , −, ∗, and ÷) into arithmetic expressions.

**The Union Operation**

Consider a query to find the names of all bank customers who have either an account or a loan or both. Note that the customer relation does not contain the information, since a customer does not need to have either an account or a loan at the bank. To answer this query, we need the information in the depositor relation and in the borrower relation.

| *customer-name* | *loan-number* |
|---|---|
| Adams | L-16 |
| Curry | L-93 |
| Hayes | L-15 |
| Jackson | L-14 |
| Jones | L-17 |
| Smith | L-11 |
| Smith | L-23 |
| Williams | L-17 |

**Figure 3.7** The *borrower* relation.

We know how to find the names of all customers with a loan in the bank:

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

We also know how to find the names of all customers with an account in the bank:

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

To answer the query, we need the **union** of these two sets; that is, we need all customer names that appear in either or both of the two relations. We find these data by the binary operation union, denoted, as in set theory, by ∪.. So the expression needed is

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

The result relation for this query appears in Figure. Notice that there are 10 tuples in the result, even though there are seven distinct borrowers and six depositors. This apparent discrepancy occurs because Smith, Jones, and Hayes are borrowers as well as depositors. Since relations are sets, duplicate values are eliminated.

| *customer-name* |
| --- |
| Adams |
| Curry |
| Hayes |
| Jackson |
| Jones |
| Smith |
| Williams |
| Lindsay |
| Johnson |
| Turner |

Observe that, in our example, we took the union of two sets, both of which consisted of customer-name values. In general, we must ensure that unions are taken between compatible relations. For example, it would not make sense to take the union of the loan relation and the borrower relation. The former is a relation of three attributes; the latter is a relation of two. Furthermore, consider a union of a set of customer names and a set of cities. Such a union would not make sense in most situations.

**The Set Difference Operation**

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

We can find all customers of the bank who have an account but not a loan by writing

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

The result relation for this query appears in Figure. As with the union operation, we must ensure that set differences are taken between compatible relations. Therefore, for a set difference operation r − s to be valid, and that the domains of the ith attribute of r and the ith attribute of s be the same.

| *customer-name* |
| --- |
| Johnson |
| Lindsay |
| Turner |

Customers with an account but no loan.

**The Rename Operation**

Unlike relations in the database, the results of relational-algebra expressions do not have a name that we can use to refer to them. It is useful to be able to give them names; the **rename** operator, denoted by the lowercase Greek letter rho ($\rho$).Given a relational-algebra expression *E*, the expression

$$\rho_x (E)$$

returns the result of expression *E* under the name *x*. A relation *r* by itself is considered a (trivial) relational-algebra expression. Thus, we can also apply the rename operation to a relation *r* to get the same relation under a newname.

| *customer-name* |
| --- |
| Adams |
| Hayes |

**Figure 3.16** Result of $\Pi_{customer\text{-}name}$ $(\sigma_{borrower.loan\text{-}number = loan.loan\text{-}number}$ $(\sigma_{branch\text{-}name = \text{“Perryridge”}} (borrower \times loan)))$

A second form of the rename operation is as follows. Assume that a relational algebra expression *E* has arity *n*. Then, the expression

$$\rho_{x(A_1,A_2,\ldots,A_n)} (E)$$

returns the result of expression E under the name x, and with the attributes renamed to A1,A2, . . .,An.

To illustrate renaming a relation, we consider the query “Find the largest account balance in the bank.” Our strategy is to (1) compute first a temporary relation consisting of those balances that are not the largest and (2) take the set difference between the relation $\Pi_{balance}$ (account) and the temporary relation just computed, to obtain the result.

Step 1: To compute the temporary relation, we need to compare the values of all account balances. We do this comparison by computing the Cartesian product **account × account** and forming a selection to compare the value of any two balances appearing in one tuple. First, we need to devise a mechanism to distinguish between the two balance attributes.We shall use the rename operation to rename one reference to the account relation; thus we can reference the relation twice without ambiguity.

| balance |
|---|
| 500 |
| 400 |
| 700 |
| 750 |
| 350 |

**Figure 3.17** Result of the subexpression $\Pi_{account.balance}\,(\sigma_{account.balance\,<\,d.balance}\,(account\,\times\,\rho_d\,(account)))$.

**Additional Operations**

The fundamental operations of the relational algebra are sufficient to express any relational-algebra query. However, if we restrict ourselves to just the fundamental operations, certain common queries are lengthy to express. Therefore, we define additional operations that do not add any power to the algebra, but simplify common queries.

**1.The Set-Intersection Operation**

The first additional-relational algebra operation that we shall define is **set intersection** (∩). Suppose that we wish to find all customers who have both a loan and anaccount. Using set intersection, we can write

$$\Pi_{customer\text{-}name}\,(borrower)\,\cap\,\Pi_{customer\text{-}name}\,(depositor)$$

Note that we can rewrite any relational algebra expression that uses set intersection by replacing the intersection operation with a pair of set-difference operations as:

$$r\,\cap\,s\,=\,r\,-\,(r\,-\,s)$$

Thus, set intersection is not a fundamental operation and does not add any power n to the relational algebra. It is simply more convenient to write r ∩ s than to write

$$r\,-\,(r\,-\,s).$$

**2. The Natural-Join Operation**

It is often desirable to simplify certain queries that require a Cartesian product. Usually, a query that involves a Cartesian product includes a selection operation on the result of the Cartesian product. Consider the query "Find the names of all customers who have a loan at the bank, along with the loan number and the loan amount." We first form the Cartesian product of the borrower and loan relations. Then, we select those tuples that pertain to only

the same loan-number, followed by the projection of the resulting customer-name, loan-number, and amount:

$$\Pi_{customer\text{-}name,\ loan.loan\text{-}number,\ amount} (\sigma_{borrower.loan\text{-}number = loan.loan\text{-}number} (borrower \times loan))$$

The natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation. It is denoted by the "join" symbol $\bowtie$ . The natural-join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes.

Although the definition of natural join is complicated, the operation is easy to apply. As an illustration, consider again the example "Find the names of all customers who have a loan at the bank, and find the amount of the loan."We express this query by using the natural join as follows:

$$\Pi_{customer\text{-}name,\ loan\text{-}number,\ amount} (borrower \bowtie loan)$$

Since the schemas for borrower and loan (that is, Borrower-schema and Loan-schema) have the attribute loan-number in common, the natural-join operation considers only pairs of tuples that have the same value on loan-number. It combines each such pair of tuples into a single tuple on the union of the two schemas (that is, customer-name, branch-name, loan-number, amount). After performing the projection, we obtain the relation in Figure 3.21.

| *customer-name* | *loan-number* | *amount* |
|---|---|---|
| Adams | L-16 | 1300 |
| Curry | L-93 | 500 |
| Hayes | L-15 | 1500 |
| Jackson | L-14 | 1500 |
| Jones | L-17 | 1000 |
| Smith | L-23 | 2000 |
| Smith | L-11 | 900 |
| Williams | L-17 | 1000 |

**Figure 3.21** Result of $\Pi_{customer\text{-}name,\ loan\text{-}number,\ amount} (borrower \bowtie loan)$

**Outer Join**

The outer-join operation is an extension of the join operation to deal with missing information. Suppose that we have the relations with the following schemas, which contain data on full-time employees:

| *employee-name* | *street* | *city* |
|---|---|---|
| Coyote | Toon | Hollywood |
| Rabbit | Tunnel | Carrotville |
| Smith | Revolver | Death Valley |
| Williams | Seaview | Seattle |

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

**Figure 3.31** The *employee* and *ft-works* relations.

Consider the employee and ft-works relations in Figure 3.31. Suppose that we

*employee* (*employee-name, street, city*)
*ft-works* (*employee-name, branch-name, salary*)

want to generate a single relation with all the information (street, city, branch name, and salary) about full-time employees. A possible approach would be to use the natural join operation as follows:

$$employee \bowtie ft\text{-}works$$

The result of this expression is as follows :-

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

**Figure 3.32** The result of $employee \bowtie ft\text{-}works$.

Notice that we have lost the street and city information about Smith, since the tuple describing Smith is absent from the ft-works relation; similarly, we have lost the branch name and salary information about Gates, since the tuple describing Gates is absent from the employee relation.

We can use the outer-join operation to avoid this loss of information. There are actually three forms of the operation: left outer join, denoted $\overset{\frown}{\bowtie}$ ; right outer join, denoted $\overset{\frown}{\bowtie}$; and full outer join, denoted $\overset{\frown}{\bowtie}$. All three forms of outer join compute the join, and add extra tuples to the result of the join. The results of the expressions appear in Figures 3.33, 3.34, and 3.35, respectively.

*employee* ⟕ *ft-works*,, *employee* ⟖ *ft-works*, and *employee* ⟗ *ft-works*

**The left outer join** ( ⟕ ) takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join

**The right outer join** ( ⟖ ) is symmetric with the left outer join: It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join.

**The full outer join**( ⟗ ) does both of those operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join. Figure 3.35 shows the result of a full outer join.

**Null Values**

Null indicates "value unknown or nonexistent," any arithmetic operations (such as ,–, ∗, /) involving null values must return a null result.

Similarly, any comparisons (such as <,<=,>,>=, =) involving a null value evaluate to special value unknown; we cannot say for sure whether the result of the comparison is true or false, so we say that the result is the new truth value unknown.

Comparisons involving nulls may occur inside Boolean expressions involving the and, or, and not operations. We must therefore define how the three Boolean operations deal with the truth value unknown.

- **and**: (*true* **and** *unknown*) = *unknown*; (*false* **and** *unknown*) = *false*; (*unknown* **and** *unknown*) = *unknown*.
- **or**: (*true* **or** *unknown*) = *true*; (*false* **or** *unknown*) = *unknown*; (*unknown* **or** *unknown*) = *unknown*.
- **not**: (**not** *unknown*) = *unknown*.

## 15.Modification of the Database

### 1.Deletion

A delete request is much the same way as a query. However, instead of displaying tuples to the user, it remove the selected tuples from the database. It can delete only whole tuples; It cannot delete values on only particular attributes.

In relational algebra a deletion is expressed by

$$r \leftarrow r - E$$

where r is a relation and E is a relational-algebra query.

**Example of relational-algebra delete requests:**

Delete all of Smith's account records.

$$depositor \leftarrow depositor - \sigma_{customer\text{-}name = \text{“Smith”}}(depositor)$$

## 2.Insertion

To insert data into a relation, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted. Obviously, the attribute values for inserted tuples must be members of the attribute's domain.

$$r \leftarrow r \cup E$$

where r is a relation and E is a relational-algebra expression. We express the insertion of a single tuple by letting E be a constant relation containing one tuple.

Suppose that we wish to insert the fact that Smith has $1200 in account A-973 at the Perryridge branch.We write

$$account \leftarrow account \cup \{(\text{A-973, “Perryridge”, 1200})\}$$
$$depositor \leftarrow depositor \cup \{(\text{“Smith”, A-973})\}$$

More generally, we might want to insert tuples on the basis of the result of a query. Suppose that we want to provide as a gift for all loan customers of the Perryridge branch a new $200 savings account. Let the loan number serve as the account number for this savings account.We write

$$r_1 \leftarrow (\sigma_{branch\text{-}name = \text{“Perryridge”}}(borrower \bowtie loan))$$
$$r_2 \leftarrow \Pi_{loan\text{-}number,\ branch\text{-}name}(r_1)$$
$$account \leftarrow account \cup (r_2 \times \{(200)\})$$
$$depositor \leftarrow depositor \cup \Pi_{customer\text{-}name,\ loan\text{-}number}(r_1)$$

## 3.Updating

In certain situations, we may wish to change a value in a tuple without changing all values in the tuple.We can use the generalized-projection operator to do this task:

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(r)$$

where each $F_i$ is either the ith attribute of r, if the ith attribute is not updated, or, if the attribute is to be updated, $F_i$ is an expression, involving only constants and the attributes of r, that gives the new value for the attribute.

## 16.Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

1. X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

**For example:**

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

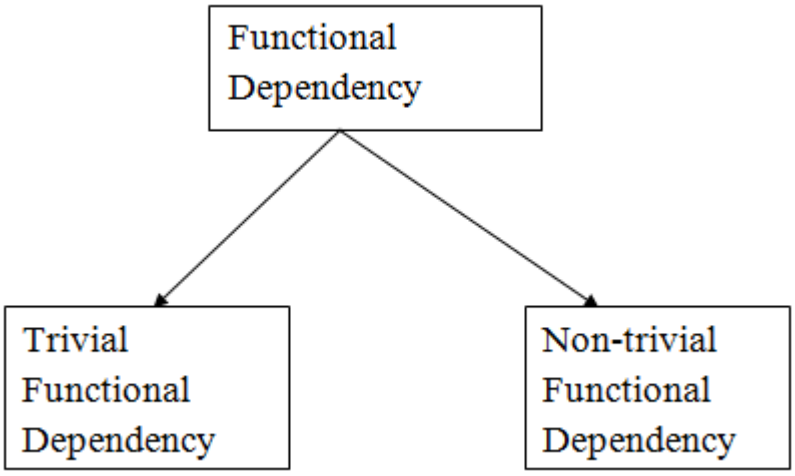
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

### Types of Functional dependency




### 1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

**Example:**

1. Consider a table with two columns Employee_Id and Employee_Name.
2. {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
3. Employee_Id is a subset of {Employee_Id, Employee_Name}.
4. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

### 2. Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

**Example:**

1. ID → Name,
2. Name → DOB