INTRODUCTION TO DBMS:

What is data?

- Data is nothing but facts and statistics stored or free flowing over a network, generally it's raw and unprocessed.
- Data becomes information when it is processed, turning it into something meaningful.
- What is database: The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently.
- It is also used to organize the data in the form of a table, schema, views, and reports, etc.
- Using the database, you can easily retrieve, insert, and delete the information.
- For example: The college Database organizes the data about the admin, staff, students and faculty etc.

What is dbms?

DBMS	File System
DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	File system is a collection of data. In this system, the user has to write the procedures for managing the database.
Searching data is easy in Dbms	Searching is difficult in File System
Dbms is structured data	Files are unstructured data
No data redundancy in Dbms	Data redundancy is there in file system
Memory utilisation well in dbms	Memory utilisation poor in file system
No data inconsistency in dbms	Inconsistency in file system

DBMS gives an abstract view of data that hides the details.	File system provides the detail of the data representation and storage of data.
DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure.	File system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will lost.
DBMS provides a good protection mechanism.	It is very difficult to protect a file under the file system.
DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	File system can't efficiently store and retrieve the data.
DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access has many problems like redirecting the file while other deleting some information or updating some information.

- A DBMS is software that allows creation, definition and manipulation of database, allowing users to store, process and analyse data easily.
- DBMS provides us with an interface or a tool, to perform various operations like creating database, storing data in it, updating data, creating tables in the database and a lot more.
- DBMS also provides protection and security to the databases.
- It also maintains data consistency in case of multiple users.

Here are some examples of popular DBMS used these days:

- MySql
- Oracle
- SQL Server
- IBM DB2

DATABASE APPLICATIONS – DBMS:

- 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.

- Industry: Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs
- Banking System: For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc.
- 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.

PURPOSE OF DATABASE SYSTEMS

- The main purpose of database systems is to manage the data. Consider a university that keeps the data of students, teachers, courses, books etc. To manage this data we need to store this data somewhere where we can add new data, delete unused data, update outdated data, retrieve data, to perform these operations on data we need a Database management system that allows us to store the data in such a way so that all these operations can be performed on the data efficiently.

Characteristics of DBMS

- Data stored into Tables: Data is never directly stored into the database. Data is stored into tables, created inside the database.
- Reduced Redundancy: In the modern world hard drives are very cheap, but earlier when hard drives were too expensive, unnecessary repetition of data in database was a big problem. But DBMS follows Normalisation which divides the data in such a way that repetition is minimum.
- Data Consistency: On Live data, i.e. data that is being continuosly updated and added, maintaining the consistency of data can become a challenge. But DBMS handles it all by itself.
- Support Multiple user and Concurrent Access: DBMS allows multiple users to work on it(update, insert, delete data) at the same time and still manages to maintain the data consistency.

- Query Language: DBMS provides users with a simple Query language, using which data can be easily fetched, inserted, deleted and updated in a database.

Advantages of DBMS

- Controls database redundancy: It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
- Data sharing: In DBMS, the authorized users of an organization can share the data among multiple users.
- Easily Maintenance: It can be easily maintainable due to the centralized nature of the database system.
- Reduce time: It reduces development time and maintenance need.
- Backup: It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.
- multiple user interface: It provides different types of user interfaces like graphical user interfaces, application program interfaces

Disadvantages of DBMS

- Cost of Hardware and Software: It requires a high speed of data processor and large memory size to run DBMS software.
- Size: It occupies a large space of disks and large memory to run them efficiently.
- Complexity: Database system creates additional complexity and requirements.
- Higher impact of failure: Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

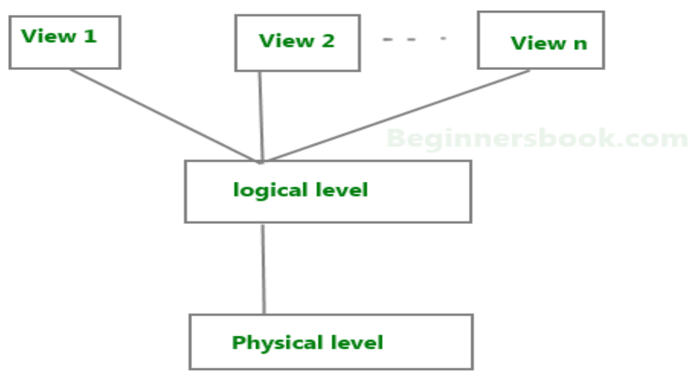
View of Data in DBMS

- 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.
- 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.

Data Abstraction in DBMS

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

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.

Logical level: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

View level: Highest level of data abstraction. This level describes the user interaction with database system.

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.

- 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.

Definition of instance:

The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

DBMS ARCHITECTURE:

- Database management systems architecture will help us understand the components of database system and the relation among them.
- The architecture of DBMS depends on the computer system on which it runs.
- 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

There are three types of DBMS architecture:

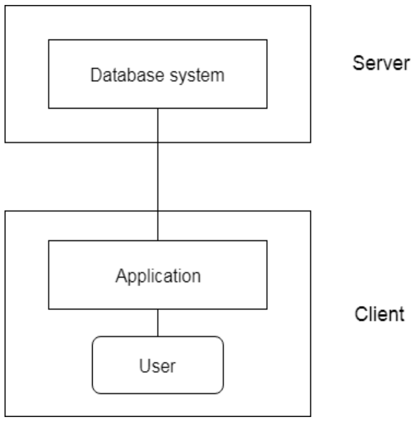
1. Single tier architecture
2. Two tier architecture
3. Three tier architecture

1-Tier Architecture

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

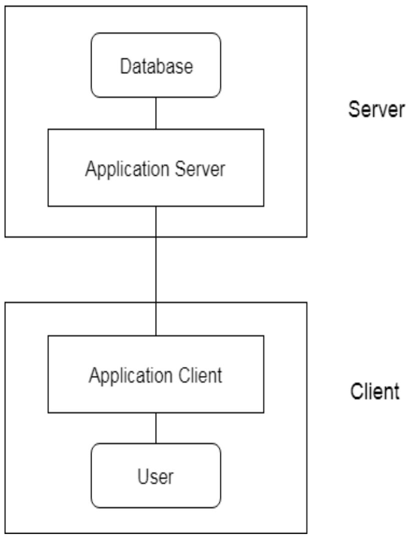
- In two-tier architecture, the Database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a reliable network.
- Whenever client machine makes a request to access the database present at server using a query language like sql, the server perform the request on the database and returns the result back to the client.
- The application connection interface such as JDBC, ODBC are used for the interaction between server and client.



3-Tier Architecture

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

communicates with server application and the server application internally communicates with the database system present at the server.



DATA MODELS:

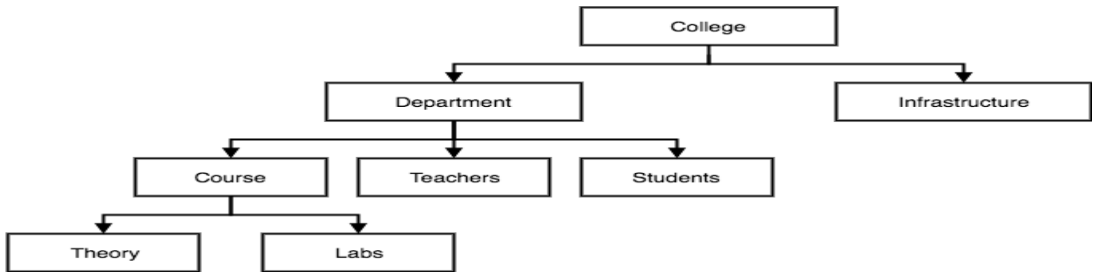
- Data Model is the modeling of the data description, data semantics, and consistency constraints of the data.
- It provides the conceptual tools for describing the design of a database at each level of data abstraction.
- Therefore, there are following four data models used for understanding the structure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

Four Types of DBMS systems are:

- Hierarchical database
- Network database
- Relational database
- ER model database

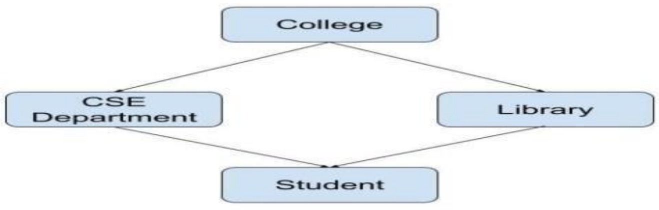
Hierarchical DBMS

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



Network Model

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



Network Model

Structure of Database Management System

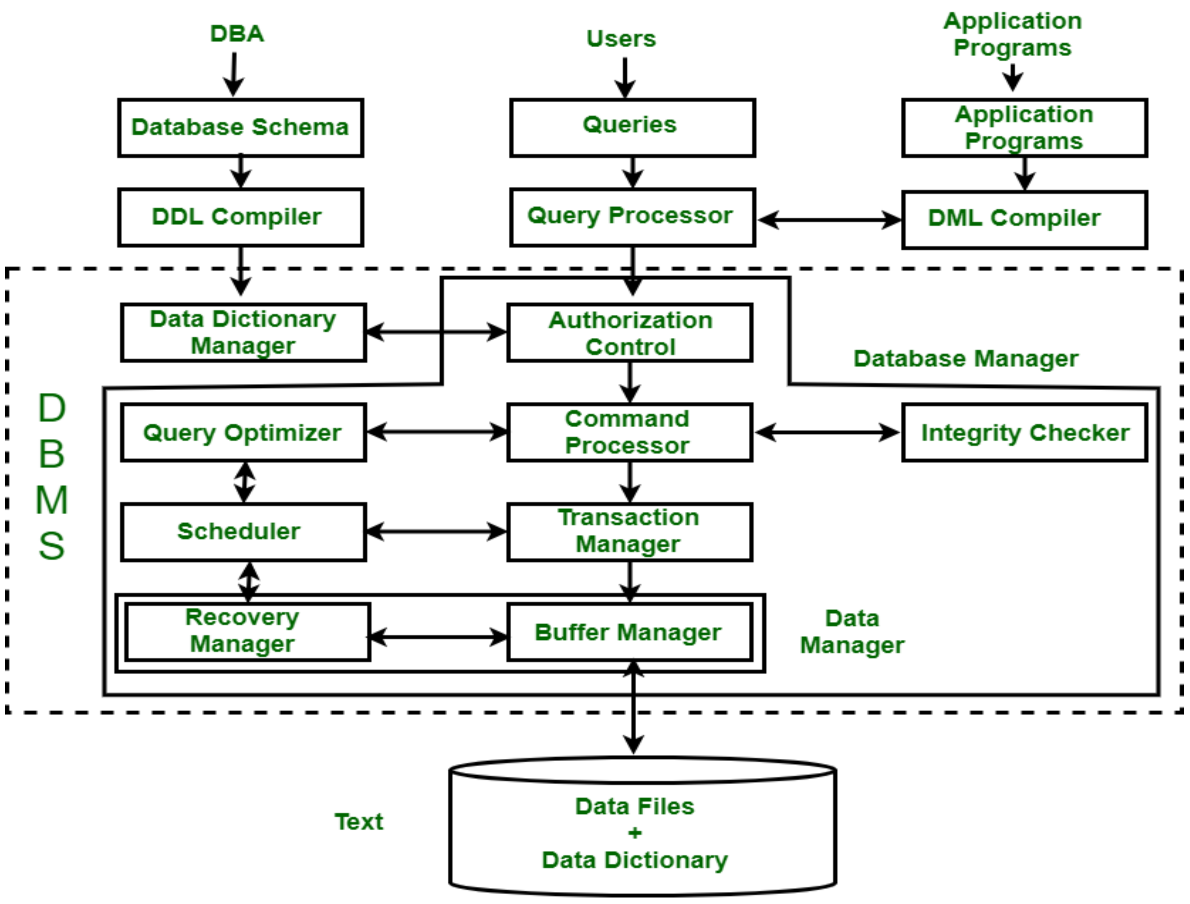
Database Management System (DBMS) is software that allows access to data stored in a database and provides an easy and effective method of –

- Defining the information.
- Storing the information.
- Manipulating the information.
- Protecting the information from system crashes or data theft.
- Differentiating access permissions for different users.

Data Theft: When somebody steals the information stored on databases, and servers, this process is known as Data Theft.

Note: *Structure of Database Management System is also referred to as Overall System Structure or Database Architecture but it is different from the tier architecture of Database.*

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



Architecture of DBMS

1. Query Processor: It interprets the requests (queries) received from end user via an application program into instructions. It also executes the user request which is received from the DML compiler.
Query Processor contains the following components –

- **DML Compiler:** It processes the DML statements into low level instruction (machine language), so that they can be executed.
- **DDL Interpreter:** It processes the DDL statements into a set of table containing meta data (data about data).
- **Embedded DML Pre-compiler:** It processes DML statements embedded in an application program into procedural calls.
- **Query Optimizer:** It executes the instruction generated by DML Compiler.

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

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

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

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

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

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

3. Disk Storage: It contains the following components –

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

- **Data Dictionary:** It contains the information about the structure of any database object. It is the repository of information that governs the metadata.
- **Indices:** It provides faster retrieval of data item.

The structure of a Database Management System (DBMS) can be divided into three main components: the Internal Level, the Conceptual Level, and the External Level.

1. Internal Level: This level represents the physical storage of data in the database. It is responsible for storing and retrieving data from the storage devices, such as hard drives or solid-state drives. It deals with low-leve implementation details such as data compression, indexing, and storage allocation.
2. Conceptual Level: This level represents the logical view of the database. It deals with the overall organization of data in the database and the relationships between them. It defines the data schema, which includes tables, attributes, and their relationships. The conceptual level is independent of any specific DBMS and can be implemented using different DBMSs.
3. External Level: This level represents the user's view of the database. It deals with how users access the dat in the database. It allows users to view data in a way that makes sense to them, without worrying about the underlying implementation details. The external level provides a set of views or interfaces to the database, which are tailored to meet the needs of specific user groups.

The three levels are connected through a schema mapping process that translates data from one level to another The schema mapping process ensures that changes made at one level are reflected in the other levels.

In addition to these three levels, a DBMS also includes a Database Administrator (DBA) component, which is responsible for managing the database system. The DBA is responsible for tasks such as database design, security management, backup and recovery, and performance tuning.

Overall, the structure of a DBMS is designed to provide a high level of abstraction to users, while still allowing low-level implementation details to be managed effectively. This allows users to focus on the logical organization of data in the database, without worrying about the physical storage or implementation details.

Relational Model in DBMS

Relational model can represent as a table with columns and rows. Each row is known as a tuple.
Each table of the column has a name or attribute.
Domain: It contains a set of atomic values that an attribute can take.
Attribute: It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai)
Relational instance: In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

Relational schema: A relational schema contains the name of the relation and name of all columns or attributes.
Relational key: In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.
Example: STUDENT Relation

NAME	ROLL_NO	PHONE_NO	ADDRESS	AGE
Ram	14795	7305758992	Noida	24
Shyam	12839	9026288936	Delhi	35
Laxman	33289	8583287182	Gurugram	20
Mahesh	27857	7086819134	Ghaziabad	27
Ganesh	17282	9028 9i3988	Delhi	40

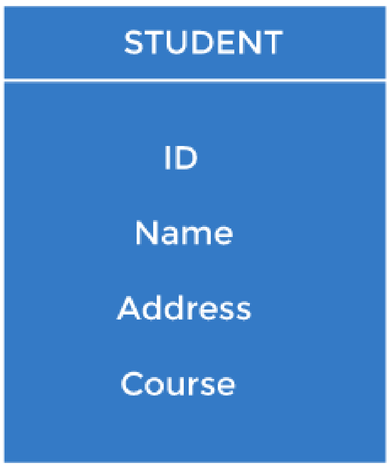
- In the given table, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.
- The instance of schema STUDENT has 5 tuples.
- t3 = <Laxman, 33289, 8583287182, Gurugram, 20>

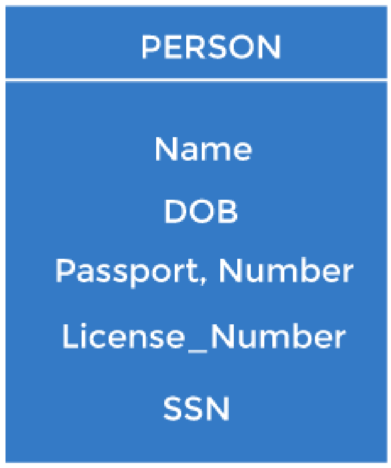
Properties of Relations

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- tuple has no duplicate value
- Order of tuple can have a different sequence

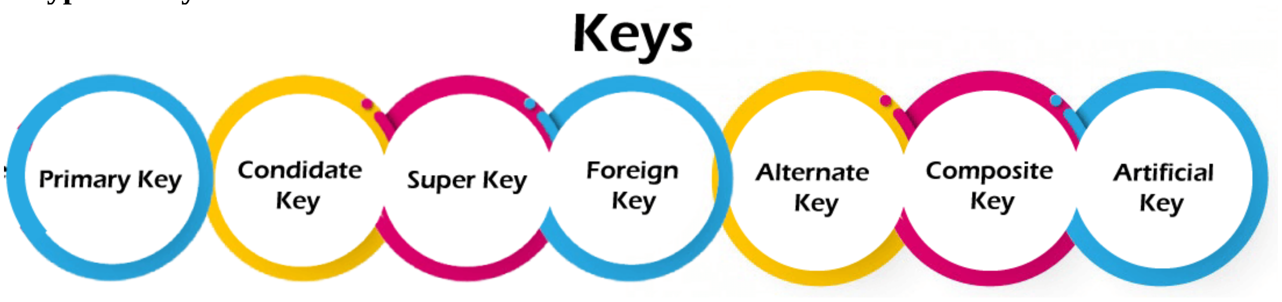
- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish an identify relationships between tables.

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



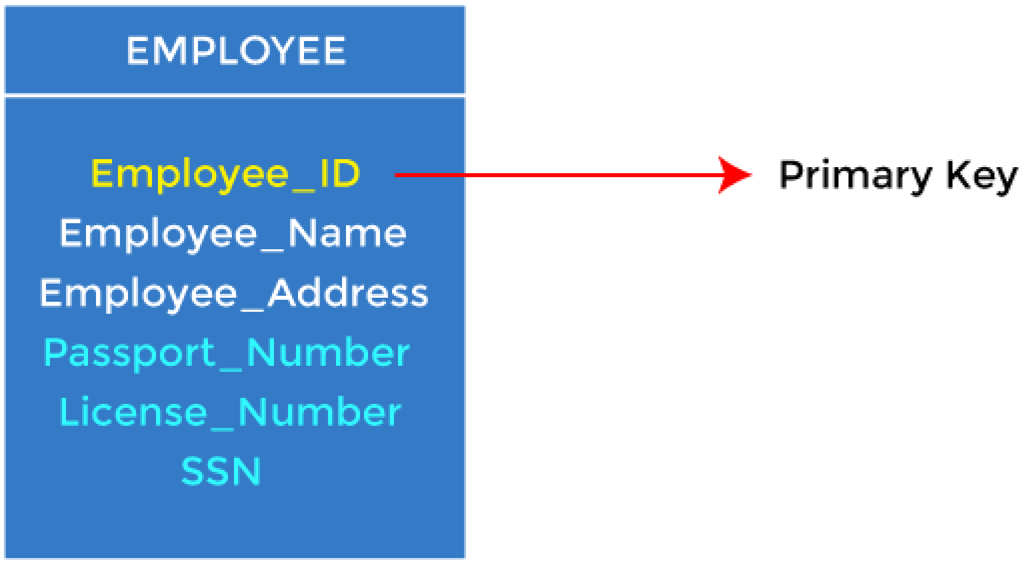


Types of keys:



1. Primary key

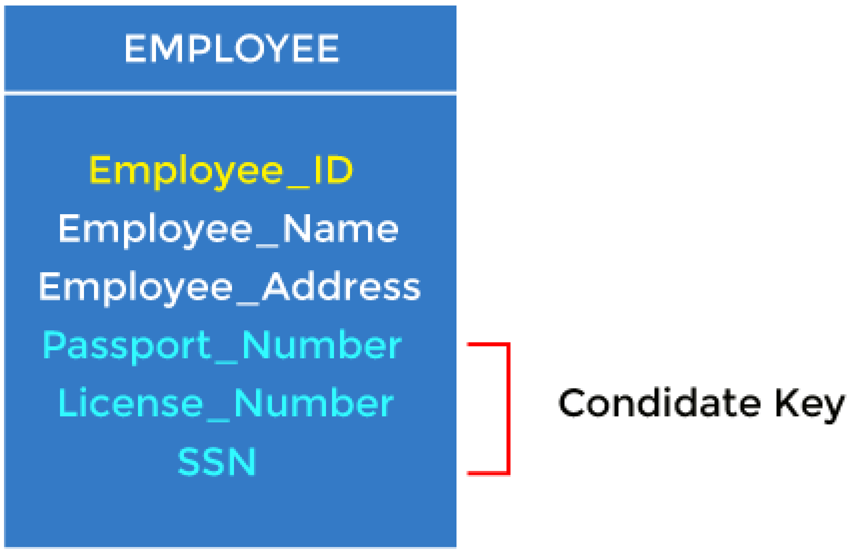
- It is the first key used to identify one and only one instance of an entity uniquely. An entity can contai multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists become a primary key.
- In the EMPLOYEE table, ID can be the primary key since it is unique for each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



2. Candidate key

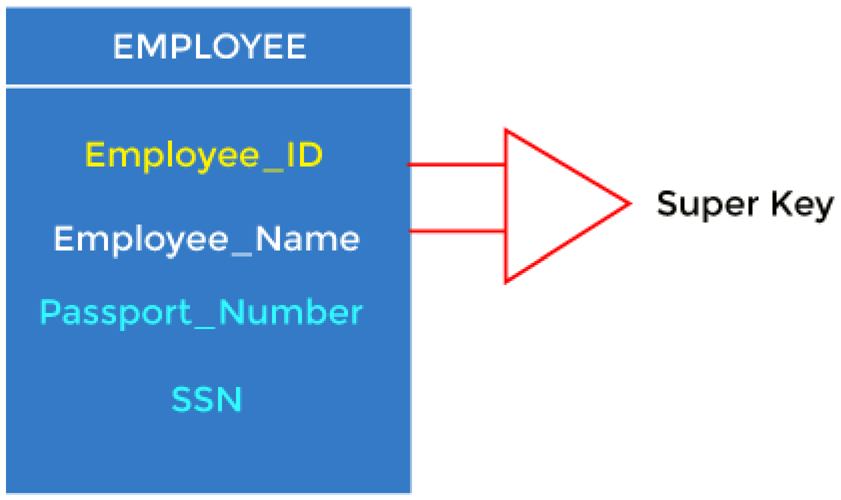
- A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
- Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

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



3. Super Key

Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.

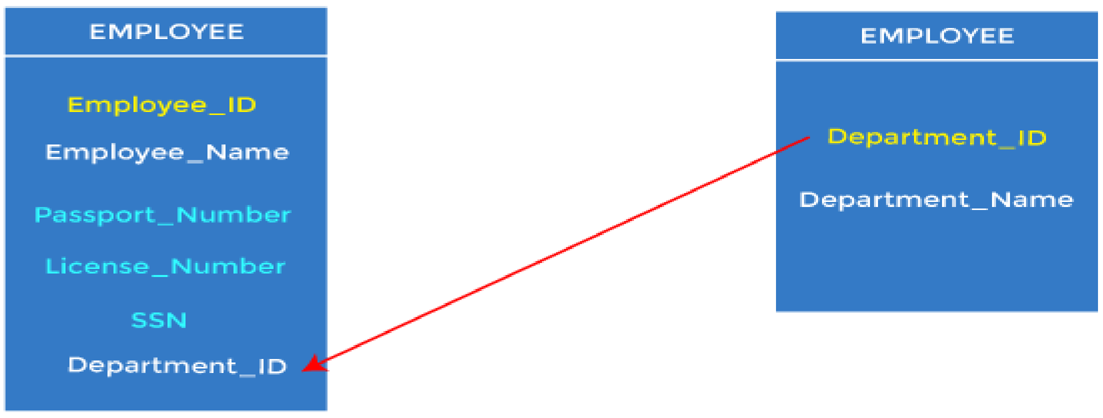


For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME), the name of tw employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be key.

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

4. Foreign key

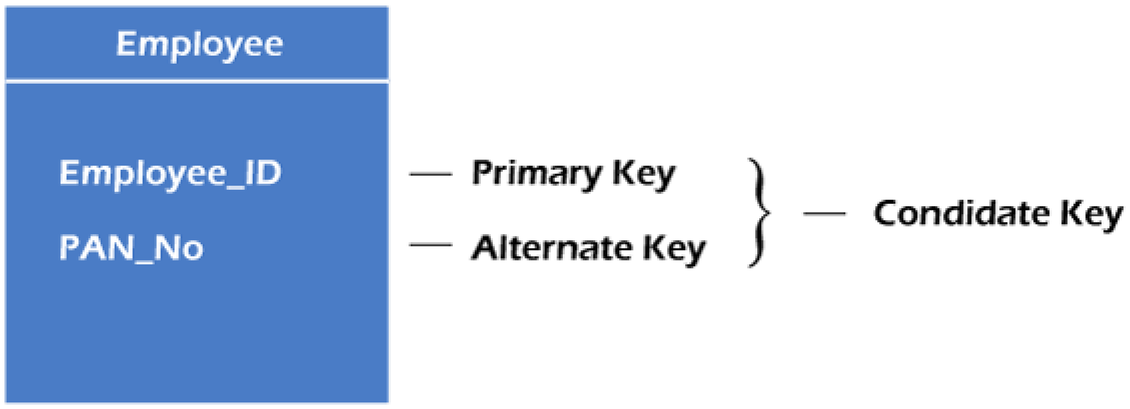
- Foreign keys are the column of the table used to point to the primary key of another table.
- Every employee works in a specific department in a company, and employee and department are tw different entities. So we can't store the department's information in the employee table. That's why w link these 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 th EMPLOYEE table.
- In the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



5. Alternate key

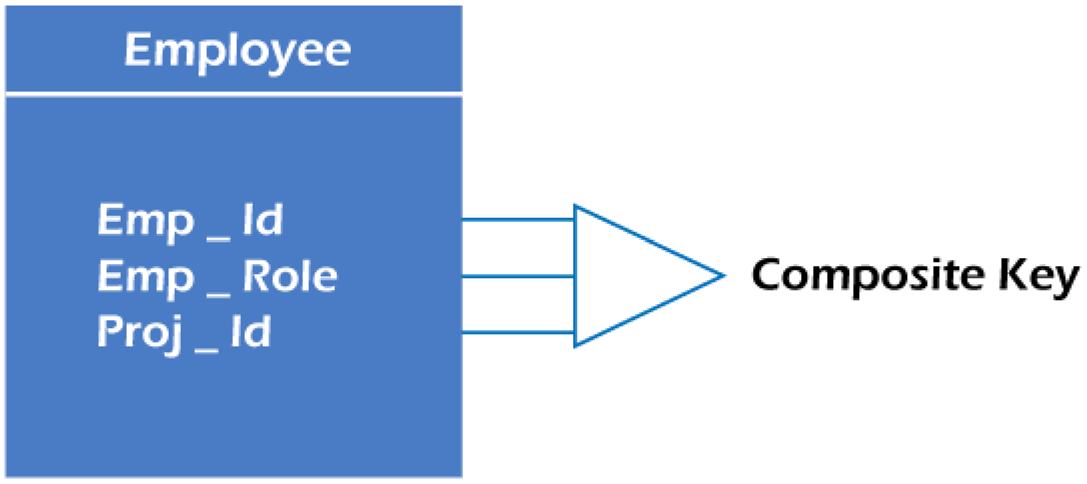
There may be one or more attributes or a combination of attributes that uniquely identify each tuple in relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen a the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternat key. **In other words,** the total number of the alternate keys is the total number of candidate keys minus th primary key. The alternate key may or may not exist. If there is only one candidate key in a relation, it does no have an alternate key.

For example, employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. I this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as th Alternate key.

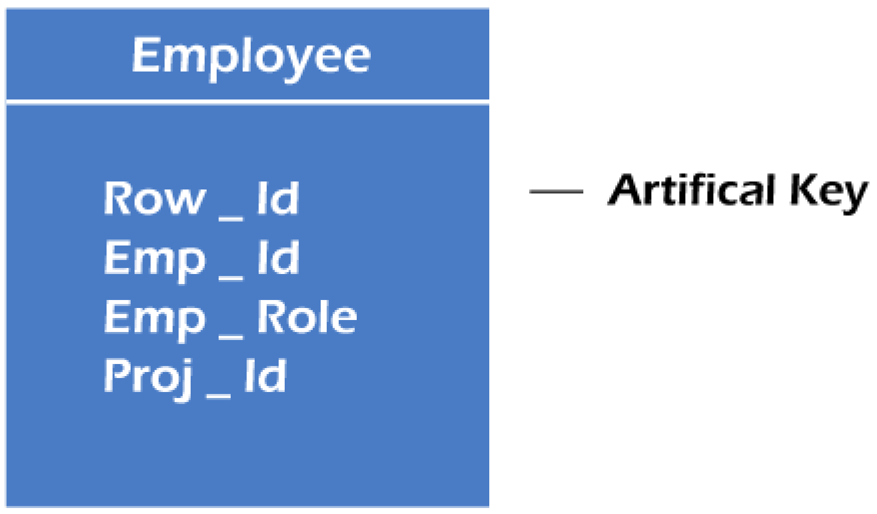


6. Composite key

Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is als known as Concatenated Key.



For example, in employee relations, we assume that an employee may be assigned multiple roles, and a employee may work on multiple projects simultaneously. So the primary key will be composed of all thre attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite ke since the primary key comprises more than one attribute.



7. Artificial key

The key created using arbitrarily assigned data are known as artificial keys. These keys are created when primary key is large and complex and has no relationship with many other relations. The data values of th artificial keys are usually numbered in a serial order.

For example, the primary key, which is composed of Emp_ID, Emp_role, and Proj_ID, is large in employe relations. So it would be better to add a new virtual attribute to identify each tuple in the relation uniquely.

Relational Algebra

Relational Algebra is a procedural query language. Relational algebra mainly provides a theoretica foundation for relational databases and SQL. The main purpose of using Relational Algebra is to defin operators that transform one or more input relations into an output relation. Given that these operators accep relations as input and produce relations as output, they can be combined and used to express potentiall complex queries that transform potentially many input relations (whose data are stored in the database) into single output relation (the query results). As it is pure mathematics, there is no use of English Keywords i Relational Algebra and operators are represented using symbols.

Fundamental Operators

These are the basic/fundamental operators used in Relational Algebra.

1. Selection(σ)
2. Projection(π)
3. Union(U)
4. Set Difference(-)
5. Set Intersection(∩)
6. Rename(ρ)
7. Cartesian Product(X)

1. Selection(σ): It is used to select required tuples of the relations.

Example:

A	**B**	**C**
1	2	4

A	**B**	**C**
2	2	3
3	2	3
4	3	4

For the above relation, **σ(c>3)R** will select the tuples which have c more than 3.

A	**B**	**C**
1	2	4
4	3	4

Note: The selection operator only selects the required tuples but does not display them. For display, the dat projection operator is used.

2. Projection(π): It is used to project required column data from a relation.

Example: Consider Table 1. Suppose we want columns B and C from Relation R.

π(B,C)R will show following columns.

B	**C**
2	4
2	3
3	4

Note: By Default, projection removes duplicate data.

3. Union(U): Union operation in relational algebra is the same as union operation in set theory.

Example:

FRENCH

Student_Name	**Roll_Number**
Ram	01
Mohan	02
Vivek	13
Geeta	17

GERMAN

Student_Name	**Roll_Number**
Vivek	13
Geeta	17
Shyam	21
Rohan	25

Consider the following table of Students having different optional subjects in their course.

π(Student_Name)FRENCH U π(Student_Name)GERMAN

Student_Name
Ram
Mohan
Vivek
Geeta
Shyam
Rohan

Note: The only constraint in the union of two relations is that both relations must have the same set o Attributes.
4. Set Difference(-): Set Difference in relational algebra is the same set difference operation as in set theory.
Example: From the above table of FRENCH and GERMAN, Set Difference is used as follows
π(Student_Name)FRENCH - π(Student_Name)GERMAN

Student_Name
Ram
Mohan

Note: The only constraint in the Set Difference between two relations is that both relations must have th same set of Attributes.
5. Set Intersection(∩): Set Intersection in relational algebra is the same set intersection operation in se theory.
Example: From the above table of FRENCH and GERMAN, the Set Intersection is used as follows
π(Student_Name)FRENCH ∩ π(Student_Name)GERMAN

Student_Name
Vivek
Geeta

Note: The only constraint in the Set Difference between two relations is that both relations must have th same set of Attributes.
6. Rename(ρ): Rename is a unary operation used for renaming attributes of a relation.
ρ(a/b)R will rename the attribute 'b' of the relation by 'a'.
7. Cross Product(X): Cross-product between two relations. Let’s say A and B, so the cross product betwee A X B will result in all the attributes of A followed by each attribute of B. Each record of A will pair wit every record of B.
Example:

A

Name	**Age**	**Sex**
Ram	14	M
Sona	15	F
Kim	20	M

B

ID	**Course**
1	DS
2	DBMS

A X B

Name	**Age**	**Sex**	**ID**	**Course**
Ram	14	M	1	DS
Ram	14	M	2	DBMS
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Kim	20	M	1	DS

Name	Age	Sex	ID	Course
Kim	20	M	2	DBMS

Note: If A has ‘n’ tuples and B has ‘m’ tuples then A X B will have ‘ n*m ‘ tuples.

Derived Operators

These are some of the derived operators, which are derived from the fundamental operators.

1. Natural Join(⋈)
2. Conditional Join

1. Natural Join(⋈): Natural join is a binary operator. Natural join between two or more relations will resul in a set of all combinations of tuples where they have an equal common attribute.

Example:

EMP

Name	ID	Dept_Name
A	120	IT
B	125	HR
C	110	Sales
D	111	IT

DEPT

Dept_Name	Manager
Sales	Y
Production	Z
IT	A

Natural join between EMP and DEPT with condition :

EMP.Dept_Name = DEPT.Dept_Name

EMP ⋈ DEPT

Name	ID	Dept_Name	Manager
A	120	IT	A
C	110	Sales	Y
D	111	IT	A

2. Conditional Join: Conditional join works similarly to natural join. In natural join, by default condition is equal between common attributes while in conditional join we can specify any condition such as greater than, less than, or not equal.
Example:

R

ID	**Sex**	**Marks**
1	F	45
2	F	55
3	F	60

S

ID	**Sex**	**Marks**
10	M	20
11	M	22
12	M	59

Join between R and S with condition **R.marks >= S.marks**

R.ID	**R.Sex**	**R.Marks**	**S.ID**	**S.Sex**	**S.Marks**
1	F	45	10	M	20
1	F	45	11	M	22
2	F	55	10	M	20
2	F	55	11	M	22
3	F	60	10	M	20
3	F	60	11	M	22
3	F	60	12	M	59

Relational Calculus

As Relational Algebra is a procedural query language, Relational Calculus is a non-procedural quer
language. It basically deals with the end results. It always tells me what to do but never tells me how to do it.
There are two types of Relational Calculus

- Tuple Relational Calculus(TRC)
- Domain Relational Calculus(DRC)
- **Tuple Relational Calculus (TRC)** is a non-procedural query language used in relational databas management systems (RDBMS) to retrieve data from tables. TRC is based on the concept of tuples, whic are ordered sets of attribute values that represent a single row or record in a database table.
- TRC is a declarative language, meaning that it specifies what data is required from the database, rathe than how to retrieve it. TRC queries are expressed as logical formulas that describe the desired tuples.
- **Syntax:** The basic syntax of TRC is as follows:
- { t | P(t) }

here t is a **tuple variable** and P(t) is a **logical formula** that describes the conditions that the tuples in th result must satisfy. The **curly braces {}** are used to indicate that the expression is a set of tuples.
For example, let's say we have a table called "Employees" with the following attributes:

Employee ID
Name
Salary
Department ID

To retrieve the names of all employees who earn more than $50,000 per year, we can use the following TR query:

{ t | Employees(t) ∧ t.Salary > 50000 }

In this query, the "Employees(t)" expression specifies that the tuple variable t represents a row in th "Employees" table. The "∧" symbol is the logical AND operator, which is used to combine the conditio "t.Salary > 50000" with the table selection.

The result of this query will be a set of tuples, where each tuple contains the Name attribute of an employe who earns more than $50,000 per year.

TRC can also be used to perform more complex queries, such as joins and nested queries, by using additiona logical operators and expressions.

While TRC is a powerful query language, it can be more difficult to write and understand than other SQL-based query languages, such as Structured Query Language (SQL). However, it is useful in certai applications, such as in the formal verification of database schemas and in academic research.
Tuple Relational Calculus is a **non-procedural query language,** unlike relational algebra. Tuple Calculu provides only the description of the query but it does not provide the methods to solve it. Thus, it explain what to do but not how to do it.

Tuple Relational Query

In Tuple Calculus, a query is expressed as

{t| P(t)}

Where t = resulting tuples
P(t) = known as Predicate and these are the conditions that are used to fetch t. Thus, it generates a set of al tuples t, such that Predicate P(t) is true for t.

P(t) may have various conditions logically combined with OR (∨), AND (∧), NOT(¬). It also uses quantifiers:
∃ t ∈ r (Q(t)) = ”there exists” a tuple in t in relation r such that predicate Q(t) is true.
∀ t ∈ r (Q(t)) = Q(t) is true “for all” tuples in relation r.

Domain Relational Calculus (DRC)

Domain Relational Calculus is similar to Tuple Relational Calculus, where it makes a list of the attributes that are to be chosen from the relations as per the conditions.
{<a1,a2,a3,.....an> | P(a1,a2,a3,.....an)}

where a1,a2,…an are the attributes of the relation and P is the condition.

Tuple Relational Calculus Examples

Table Customer

Customer name	Street	City
Saurabh	A7	Patiala
Mehak	B6	Jalandhar
Sumiti	D9	Ludhiana
Ria	A5	Patiala

Table Branch

Branch name	Branch City
ABC	Patiala
DEF	Ludhiana
GHI	Jalandhar

Table Account

Account number	Branch name	Balance
1111	ABC	50000
1112	DEF	10000
1113	GHI	9000

Account number	Branch name	Balance
1114	ABC	7000

1. **Table Loan**

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L49	GHI	9000
L98	DEF	65000

Table Borrower

Customer name	Loan number
Saurabh	L33
Mehak	L49
Ria	L98

Table Depositor

Customer name	Account number
Saurabh	1111
Mehak	1113
Suniti	1114

Example 1: Find the loan number, branch, and amount of loans greater than or equal to 10000 amount.

{t| t ∈ loan ∧ t[amount]>=10000}

Resulting relation:

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000

Loan number	Branch name	Amount
L98	DEF	65000

In the above query, t[amount] is known as a tuple variable.

Example 2: Find the loan number for each loan of an amount greater or equal to 10000.

2. {t| ∃ s ∈ loan(t[loan number] = s[loan number]

   ∧ s[amount]>=10000)}

Resulting relation:

Loan number
L33
L35
L98

Example 3: Find the names of all customers who have a loan and an account at the bank.

{t | ∃ s ∈ borrower(t[customer-name] = s[customer-name])

∧ ∃ u ∈ depositor(t[customer-name] = u[customer-name])}

Resulting relation:

Customer name
Saurabh
Mehak

Example 4: Find the names of all customers having a loan at the "ABC" branch.

{t | ∃ s ∈ borrower(t[customer-name] = s[customer-name]

∧ ∃ u ∈ loan(u[branch-name] = "ABC" ∧ u[loan-number] = s[loan-number]))}

Resulting relation:

Customer name
Saurabh

Entity-Relationship Model

Introduction of ER Model

The Entity Relational Model is a model for identifying entities to be represented in the database an representation of how those entities are related. The ER data model specifies enterprise schema that represent the overall logical structure of a database graphically.
The Entity Relationship Diagram explains the relationship among the entities present in the database. E models are used to model real-world objects like a person, a car, or a company and the relation between thes real-world objects. In short, ER Diagram is the structural format of the database.

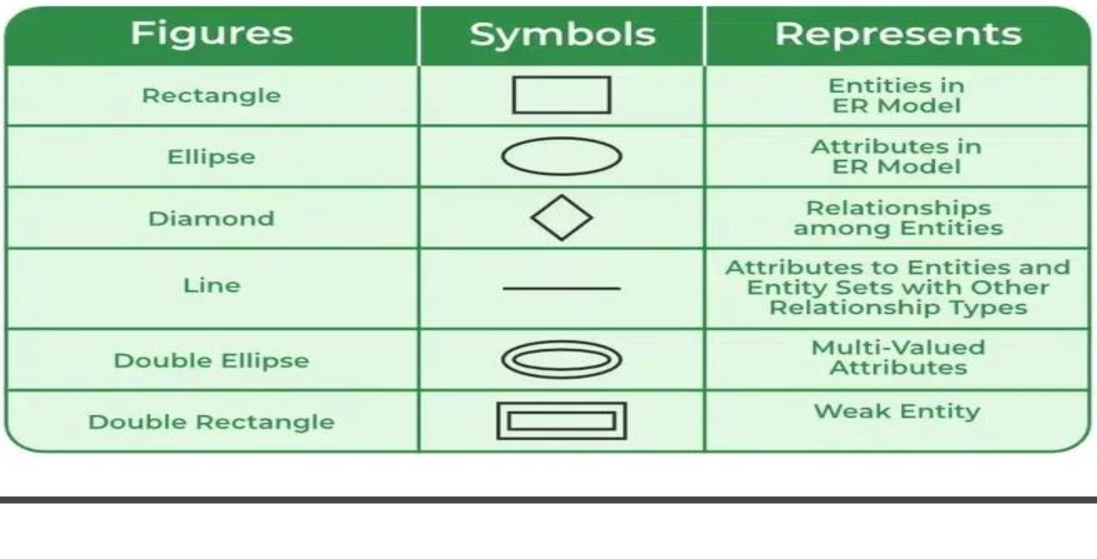
Why Use ER Diagrams In DBMS?

- ER diagrams are used to represent the E-R model in a database, which makes them easy to be converted int relations (tables).
- ER diagrams provide the purpose of real-world modeling of objects which makes them intently useful.
- ER diagrams require no technical knowledge and no hardware support.
- These diagrams are very easy to understand and easy to create even for a naive user.
- It gives a standard solution for visualizing the data logically.

Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which consists of thes symbols:

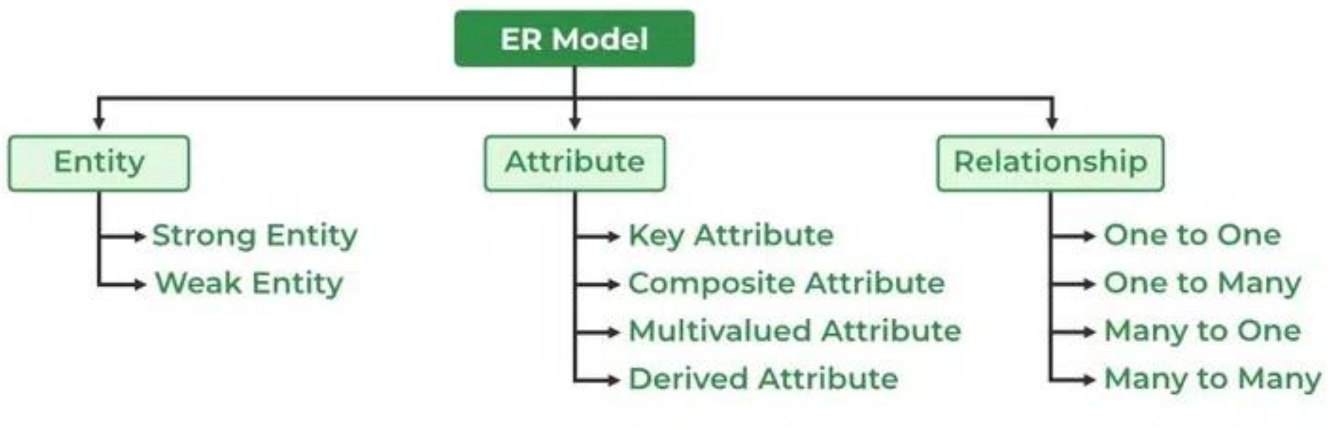
- **Rectangles:** Rectangles represent Entities in ER Model.
- **Ellipses:** Ellipses represent Attributes in ER Model.
- **Diamond:** Diamonds represent Relationships among Entities.
- **Lines:** Lines represent attributes to entities and entity sets with other relationship types.
- **Double Ellipse:** Double Ellipses represent Multi-Valued Attributes.
- **Double Rectangle:** Double Rectangle represents a Weak Entity.



Figures	Symbols	Represents
Rectangle	▭	Entities in ER Model
Ellipse	⬭	Attributes in ER Model
Diamond	◇	Relationships among Entities
Line	—	Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse	◎	Multi-Valued Attributes
Double Rectangle	⧈	Weak Entity

Symbols used in ER Diagram

Components of ER Diagram
ER Model consists of Entities, Attributes, and Relationships among Entities in a Database System.

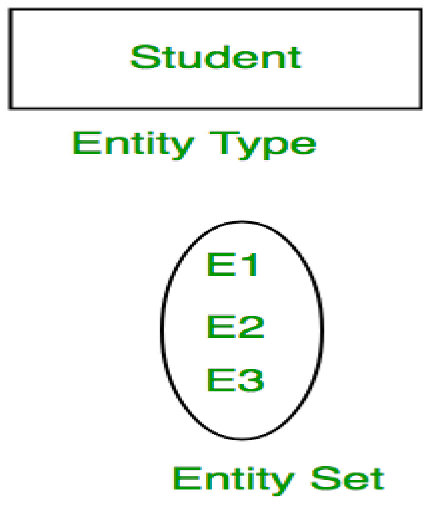


Components of ER Diagram

Entity

An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it ma
be an object with a conceptual existence – a company, a job, or a university course.

Entity Set: An Entity is an object of Entity Type and a set of all entities is called an entity set. For Example, E
is an entity having Entity Type Student and the set of all students is called Entity Set. In ER diagram, Entit
Type is represented as:



Entity Set

1. Strong Entity

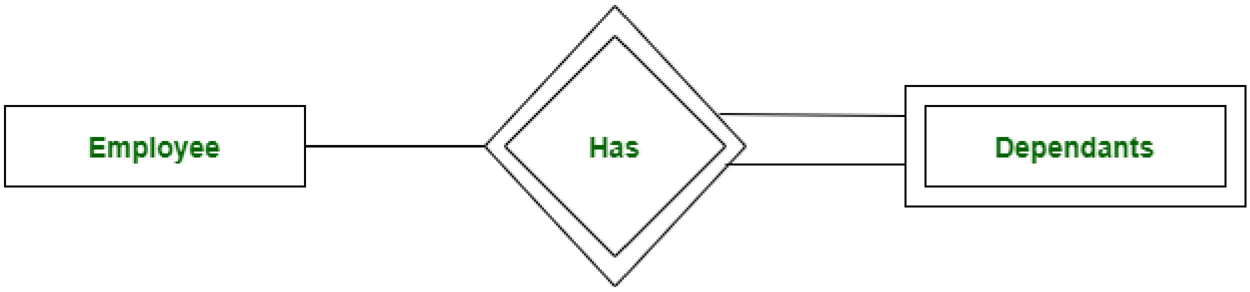
A Strong Entity is a type of entity that has a key Attribute. Strong Entity does not depend on other Entity in th Schema. It has a primary key, that helps in identifying it uniquely, and it is represented by a rectangle. Thes are called Strong Entity Types.

2. Weak Entity

An Entity type has a key attribute that uniquely identifies each entity in the entity set. But some entity type exists for which key attributes can't be defined. These are called Weak Entity types.

For Example, A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents don't have existed without the employee. So Dependent will be a **Weak Entity Type** and Employee will be Identifying Entity type for Dependent, which means it is **Strong Entity Type**.

A weak entity type is represented by a Double Rectangle. The participation of weak entity types is always total. The relationship between the weak entity type and its identifying strong entity type is called identifying relationship and it is represented by a double diamond.



Strong Entity and Weak Entity

Attributes

Attributes are the properties that define the entity type. For example, Roll_No, Name, DOB, Age, Address, an Mobile_No are the attributes that define entity type Student. In ER diagram, the attribute is represented by a oval.

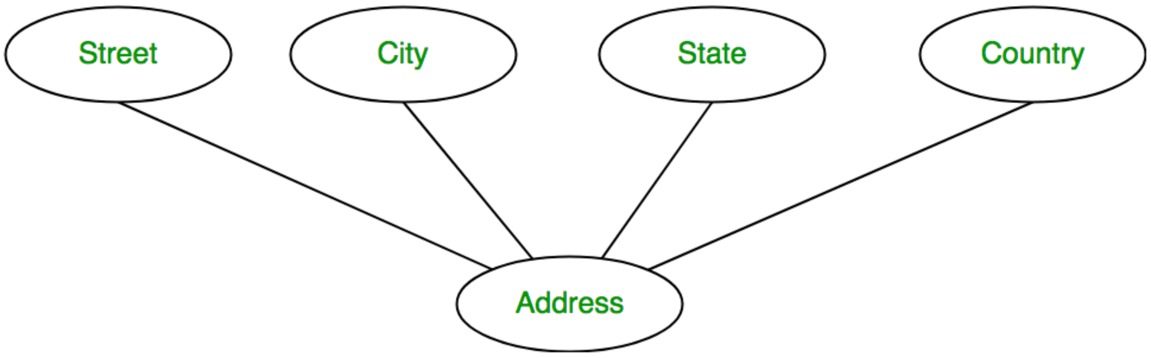
Attribute

1. Key Attribute

The attribute which **uniquely identifies each entity** in the entity set is called the key attribute. For exampl Roll_No will be unique for each student. In ER diagram, the key attribute is represented by an oval wit underlying lines.

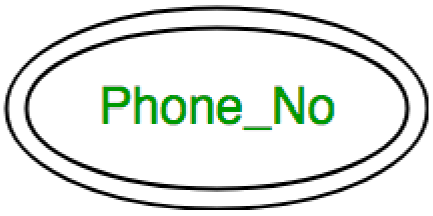
Key Attribute

2. Composite Attribute
An attribute **composed of many other attributes** is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country. In ER diagram, the composite attribute is represented by an oval comprising of ovals.



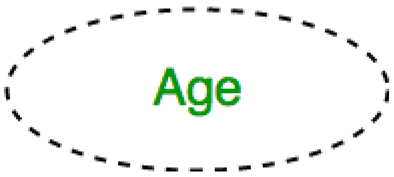
Composite Attribute

3. Multivalued Attribute
An attribute consisting of more than one value for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval.



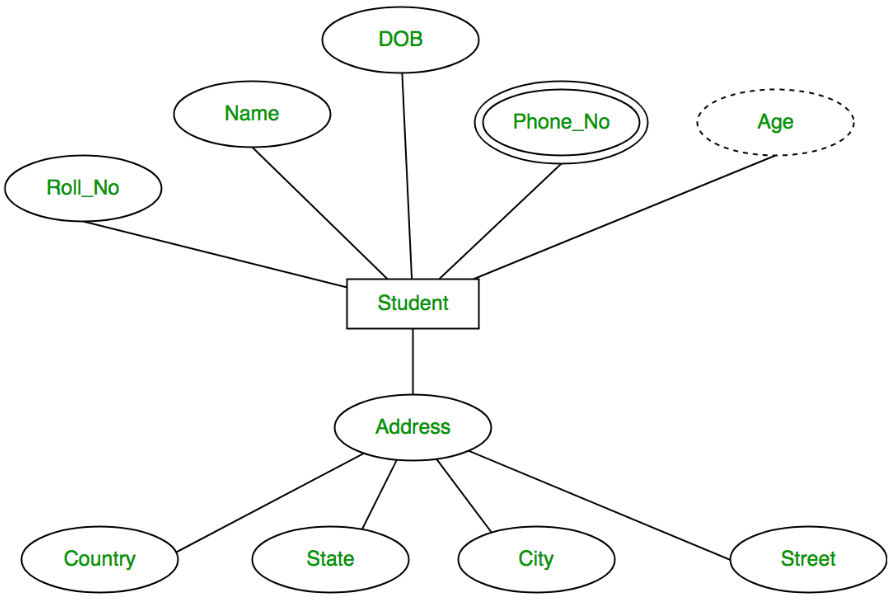
Multivalued Attribute

4. Derived Attribute
An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, the derived attribute is represented by a dashed oval.



Derived Attribute

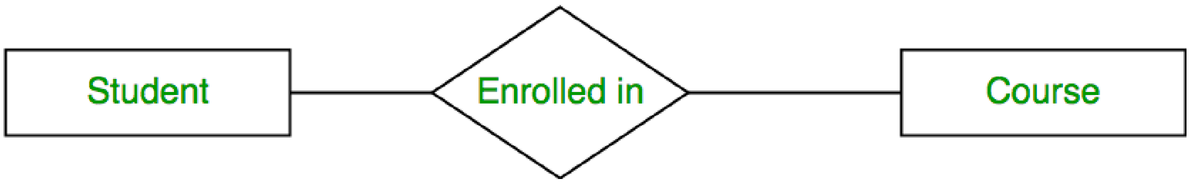
The Complete Entity Type Student with its Attributes can be represented as:



Entity and Attributes

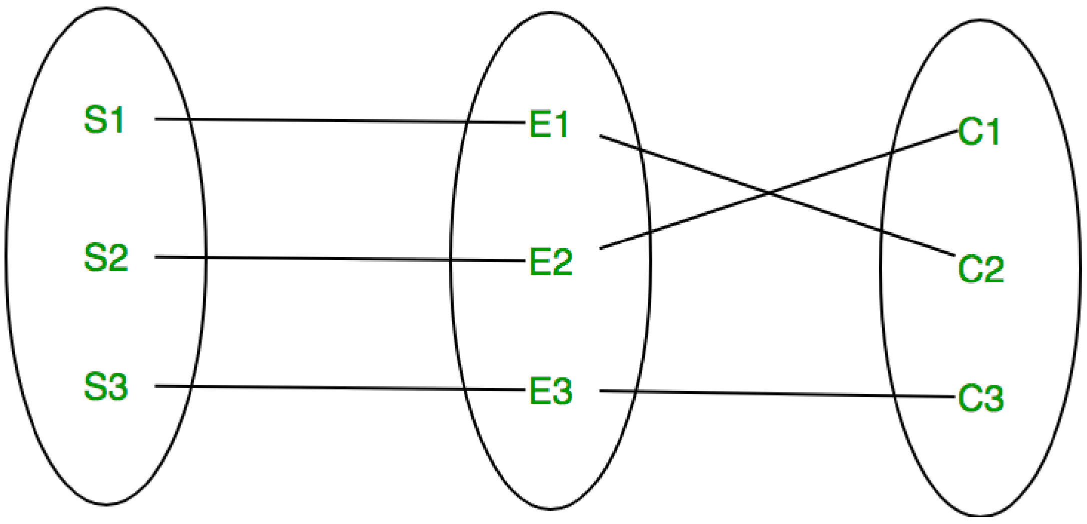
Relationship Type and Relationship Set

A Relationship Type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, the relationship type is represented by a diamond and connecting the entities with lines.



Entity-Relationship Set

A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 as enrolled in C2, S2 as enrolled in C1, and S3 as registered in C3.

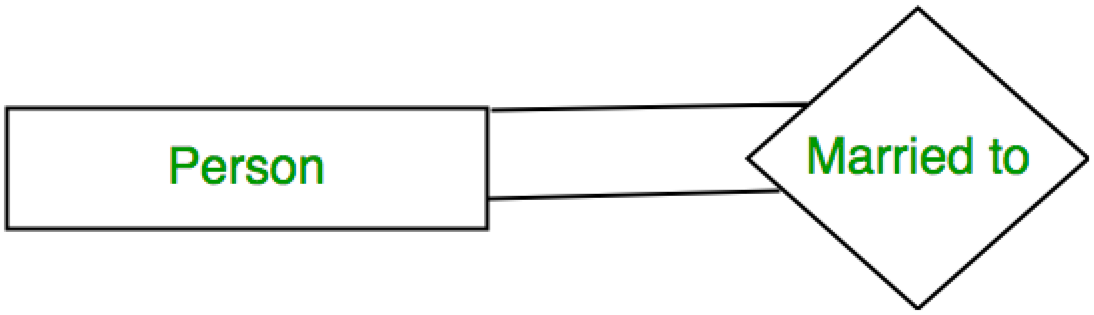


Relationship Set

Degree of a Relationship Set

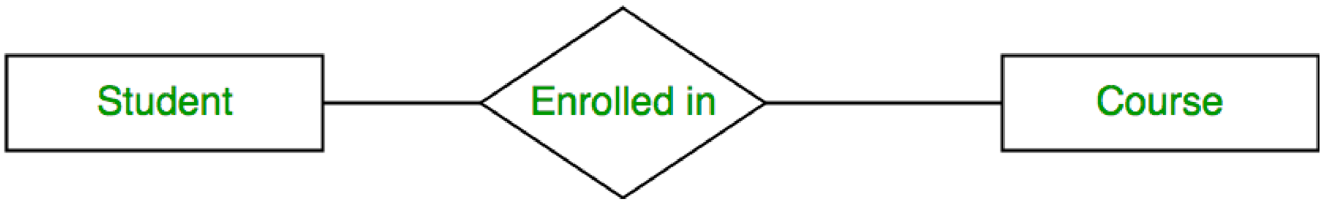
The number of different entity sets participating in a relationship set is called the degree of a relationship set.

1. Unary Relationship: When there is only ONE entity set participating in a relation, the relationship is calle a unary relationship. For example, one person is married to only one person.



Unary Relationship

3. **Binary Relationship:** When there are TWO entities set participating in a relationship, the relationship i called a binary relationship. For example, a Student is enrolled in a Course.
4.



Binary Relationship

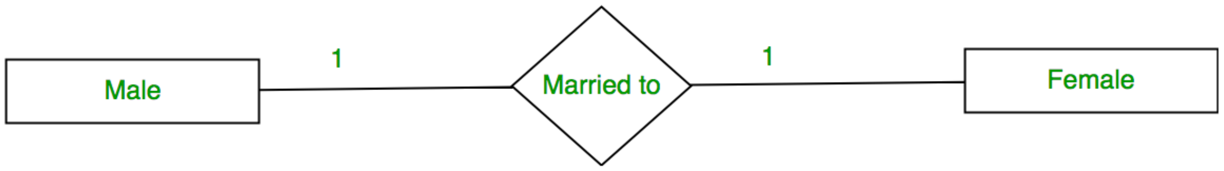
3. n-ary Relationship: When there are n entities set participating in a relation, the relationship is called an n-ary relationship.

Cardinality

The number of times an entity of an entity set participates in a relationship set is known as cardinality. Cardinality can be of different types:

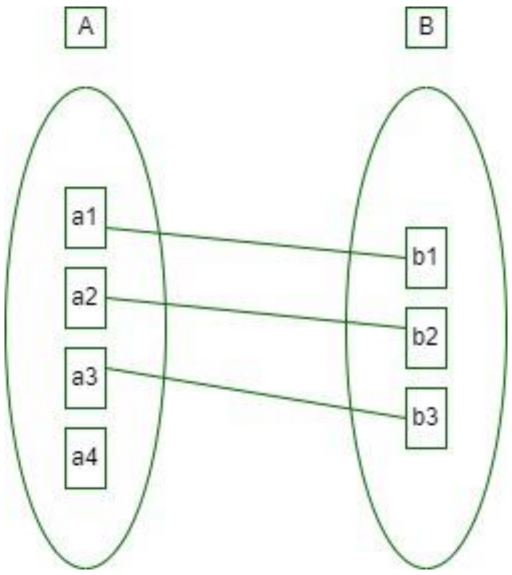
1. One-to-One: When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one. Let us assume that a male can marry one female and a female can marry one male. So the relationship will be one-to-one.

the total number of tables that can be used in this is 2.



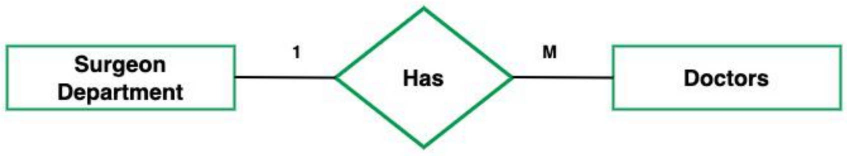
One-to-One Cardinality

Using Sets, it can be represented as:



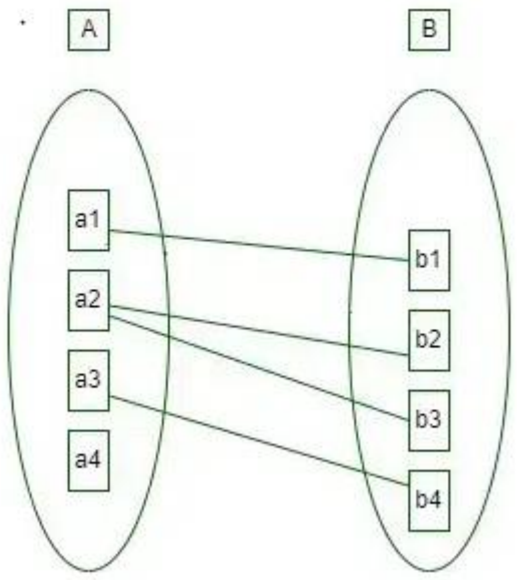
Set Representation of One-to-One

2. One-to-Many: In one-to-many mapping as well where each entity can be related to more than one relationship and the total number of tables that can be used in this is 2.



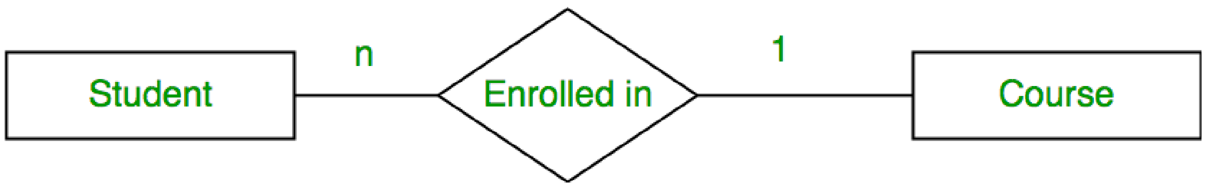
One to Many

Using sets, one-to-many cardinality can be represented as:



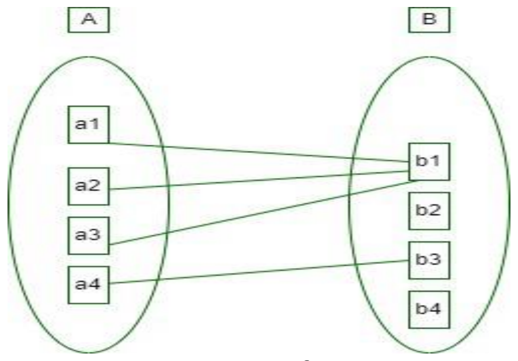
Set Representation of One-to-Many

3. Many-to-One: When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course. The total number of tables that can be used in this is 3.



Many-to-One Relationship

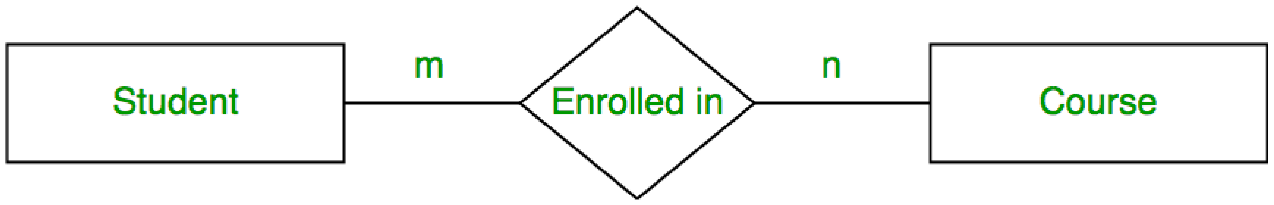
Using Sets, it can be represented as:



Set Representation of Many-to-One

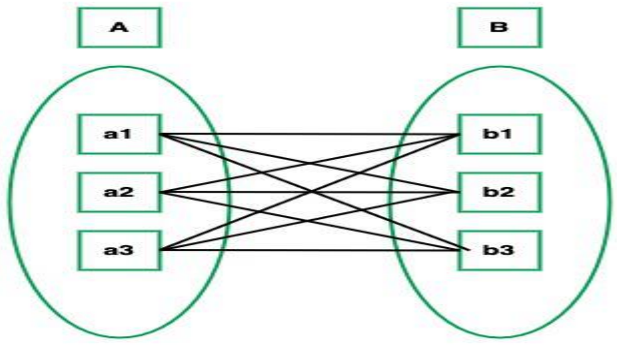
In this case, each student is taking only 1 course but 1 course has been taken by many students.

4. Many-to-Many: When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.
the total number of tables that can be used in this is 3.



Many-to-Many

Using Sets, it can be represented as:

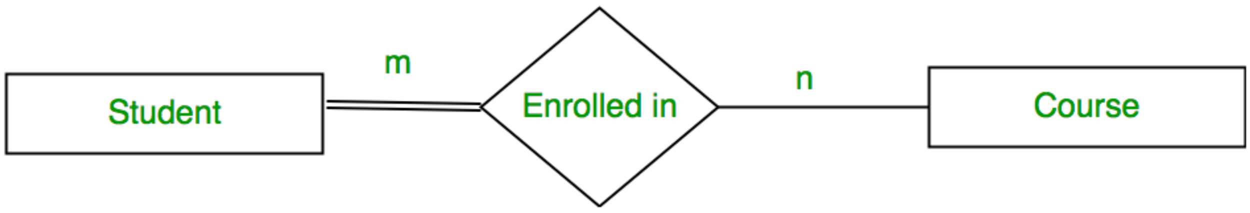


Many-to-Many Set Representation

In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3, and S4. So it is many-to-many relationships.

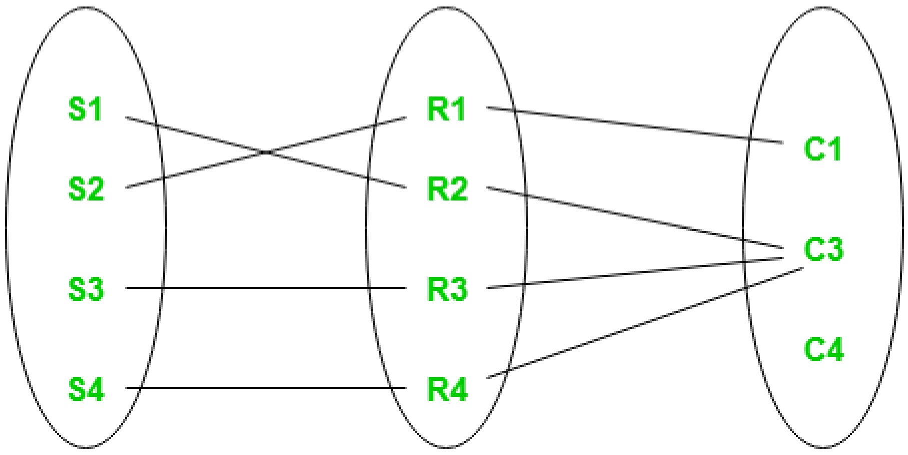
Participation Constraint
Participation Constraint is applied to the entity participating in the relationship set.

1. Total Participation – Each entity in the entity set must participate in the relationship. If each student must enroll in a course, the participation of students will be total. Total participation is shown by a double line in the ER diagram.
2. Partial Participation – The entity in the entity set may or may NOT participate in the relationship. If some courses are not enrolled by any of the students, the participation in the course will be partial.
The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



Total Participation and Partial Participation

Using Set, it can be represented as,



Set representation of Total Participation and Partial Participation

Every student in the Student Entity set participates in a relationship but there exists a course C4 that is not taking part in the relationship.

How to Draw ER Diagram?

- The very first step is Identifying all the Entities, and place them in a Rectangle, and labeling them accordingly.
- The next step is to identify the relationship between them and pace them accordingly using the Diamond, and make sure that, Relationships are not connected to each other.
- Attach attributes to the entities properly.
- Remove redundant entities and relationships.
- Add proper colors to highlight the data present in the database.

Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

DBMS – ER Design Issues

Here are some of the issues that can occur while ER diagram design process:

1. Choosing Entity Set vs Attributes

Here we will discuss how choosing an entity set vs an attribute can change the whole ER design semantics. To understand this lets take an example, let’s say we have an entity set Student with attributes such as student-name and student-id. Now we can say that the student-id itself can be an entity with the attributes like student-class and student-section.

Now if we compare the two cases we discussed above, in the first case we can say that the student can have only one student id, however in the second case when we chose student id as an entity it implied that a student can have more than one student id.

2. Choosing Entity Set vs. Relationship Sets

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

Let’s take an example to understand it better: A person takes a loan from a bank, here we have two entities person and bank and their relationship is loan. This is fine until there is a need to disburse a joint loan, in such case a new relationship needs to be created to define the relationship between the two individuals who have taken joint loan. In this scenario, it is better to choose loan as an entity set rather than a relationship set.

3. Choosing Binary vs n-ary Relationship Sets

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

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

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

4. Placing Relationship Attributes

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

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

Extended ER Features

Extended ER is a high-level data model that incorporates the extensions to the original ER model. Enhanced ER models are high level models that represent the requirements and complexities of complex databases.

The extended Entity Relationship (ER) models are three types as given below −

- Aggregation
- Specialization
- Generalization

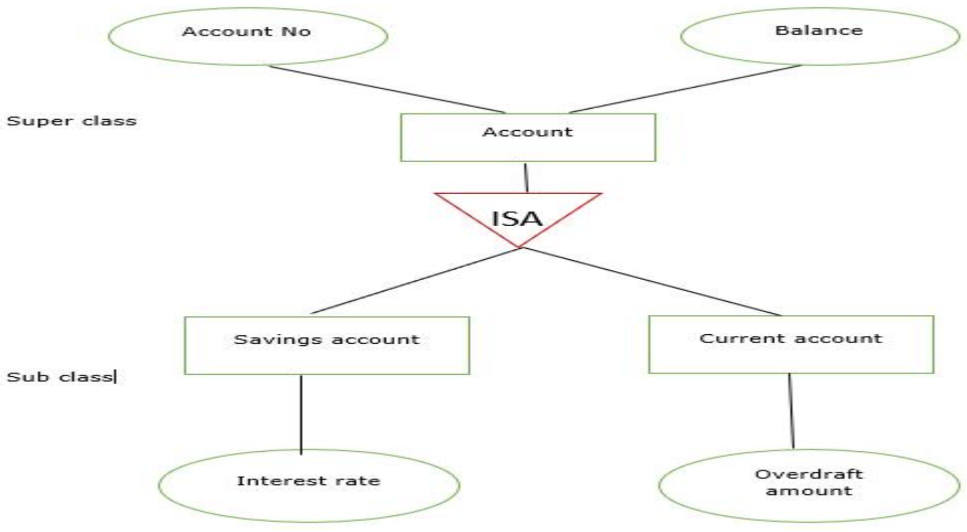
Specialization

The process of designing sub groupings within an entity set is called specialization. It is a top-down process. If an entity set is given with all the attributes in which the instances of the entity set are differentiated according to the given attribute value, then that sub-classes or the sub-entity sets can be formed from the given attribute.

Example

Specialization of a person allows us to distinguish a person according to whether they are employees or customers. Specialization of account creates two entity sets: savings account and current account.

In the E-R diagram specialization is represented by triangle components labeled ISA. The ISA relationship is referred as superclass- subclass relationship as shown below −



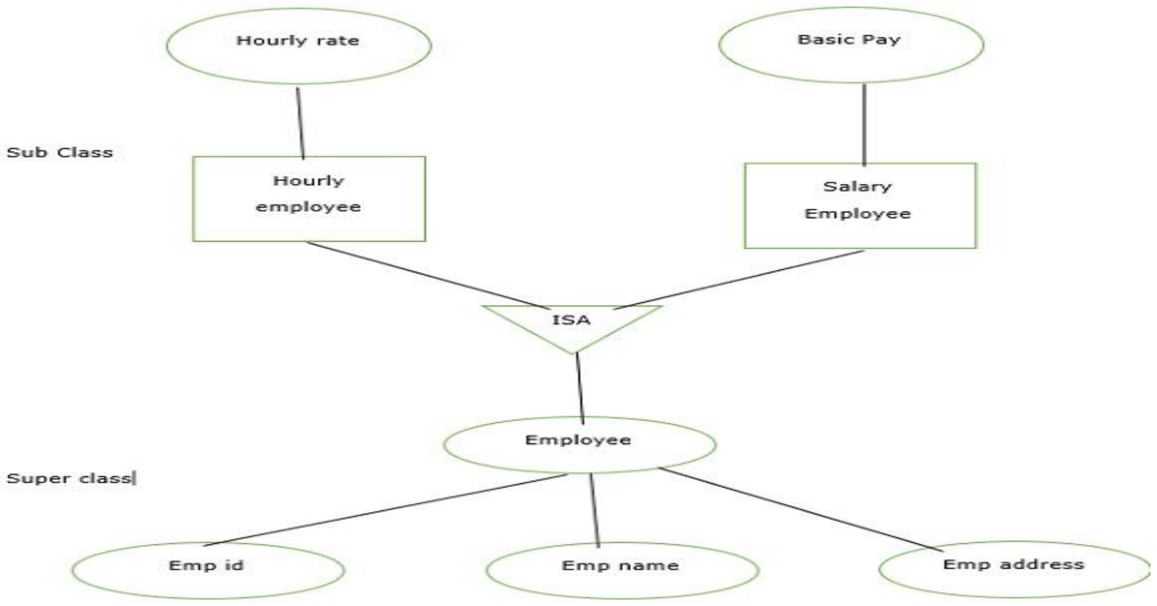
Generalization

It is the reverse process of specialization. It is a bottom-up approach.

It converts subclasses to superclasses. This process combines a number of entity sets that share the same features into higher-level entity sets.

If the sub-class information is given for the given entity set then, ISA relationship type will be used to represent the connectivity between the subclass and superclass as shown below −

Example

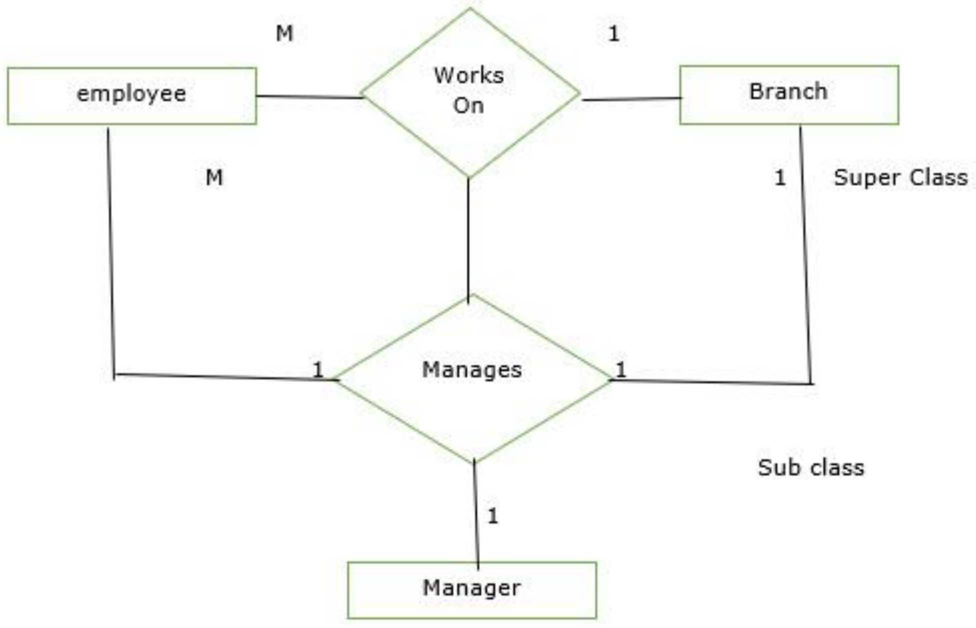


Aggregation

It is an abstraction in which relationship sets are treated as higher level entity sets and can participate i relationships. Aggregation allows us to indicate that a relationship set participates in another relationship set.

Aggregation is used to simplify the details of a given database where ternary relationships will be changed int binary relationships. Ternary relation is only one type of relationship which is working between three entities.

Aggregation is shown in the image below −

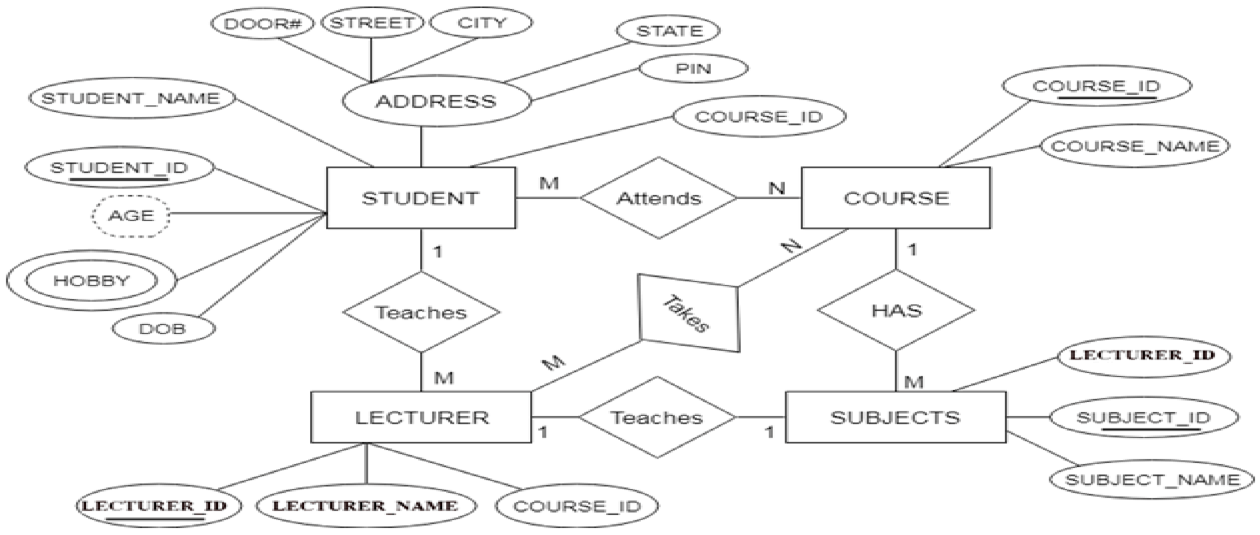


Reduction of ER diagram to Table

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

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

The ER diagram is given below:



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

- **Entity type becomes a table.**

In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.

- **All single-valued attribute becomes a column for the table.**

In the STUDENT entity, STUDENT_NAME and STUDENT_ID form the column of STUDENT tabl 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 ke 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 single column of STUDENT table. Hence we create a table STUD_HOBBY with column name STUDENT_I 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#, STREE 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 th 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 th tables. Table structure for the given ER diagram is as below:

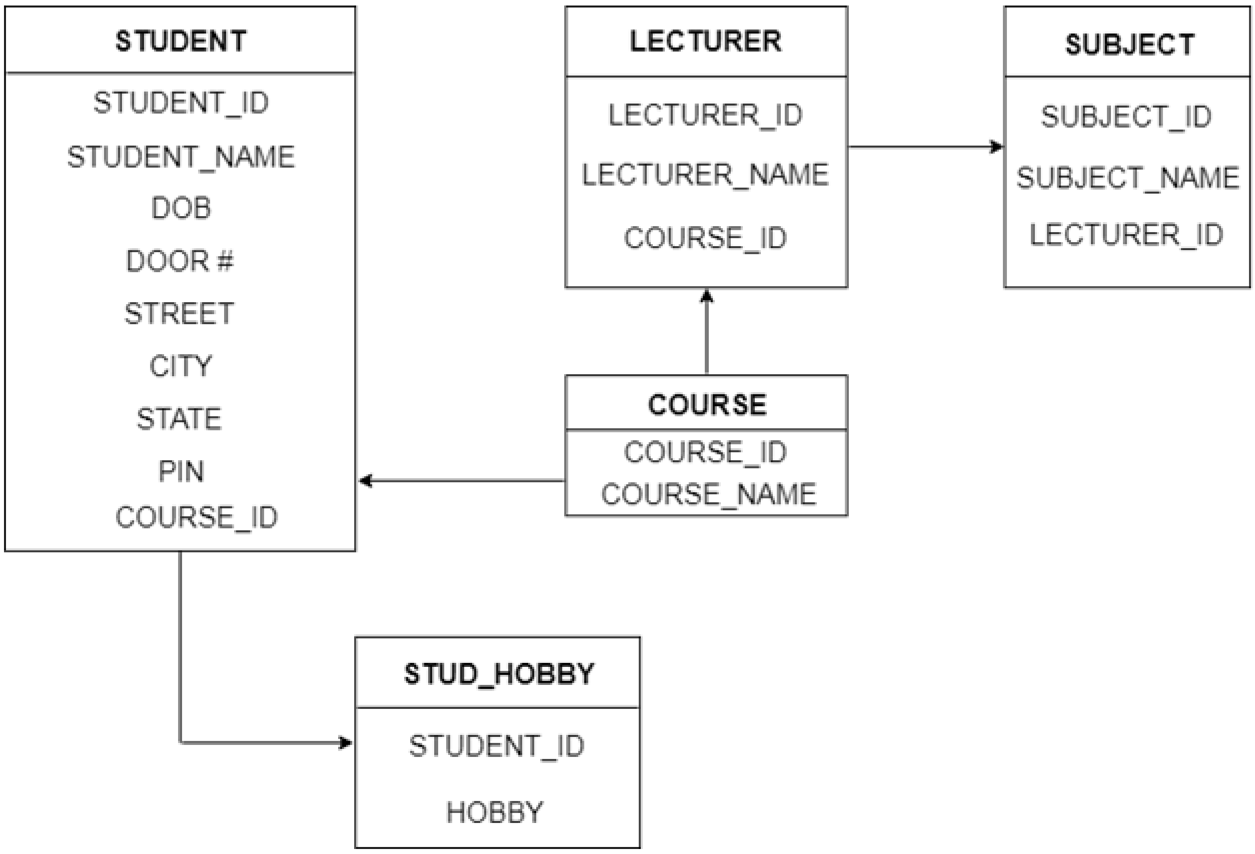


Figure: Table structure