**UNIT - II Introduction to the Relational Model: Integrity constraint over relations, enforcing integrity constraints, querying relational data, logical data base design, introduction to views, destroying/altering tables and views. Relational Algebra, Tuple relational Calculus, Domain relational calculus.**

Database Languages in DBMS

* A DBMS has appropriate languages and interfaces to express database queries and updates.
* Database languages can be used to read, store and update the data in the database.

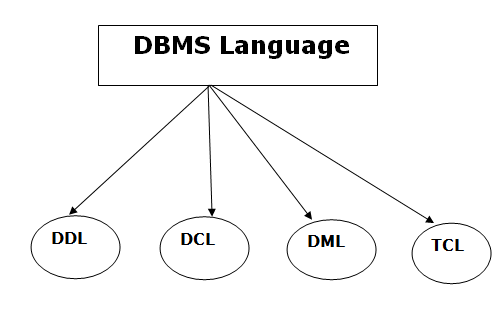
Types of Database Languages

Data Definition Language (DDL)

Data Manipulation Language (DML)

Data Control Language (DCL)

Transaction Control Language (TCL)



1. Data Definition Language (DDL)

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

Here are some tasks that come under DDL:

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

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

2. Data Manipulation Language (DML)

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

Here are some tasks that come under DML:

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

3. Data Control Language (DCL)

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

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

Here are some tasks that come under DCL:

* **Grant:** It is used to give user access privileges to a database.
* **Revoke:** It is used to take back permissions from the user.

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

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

4. Transaction Control Language (TCL)

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

Here are some tasks that come under TCL:

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

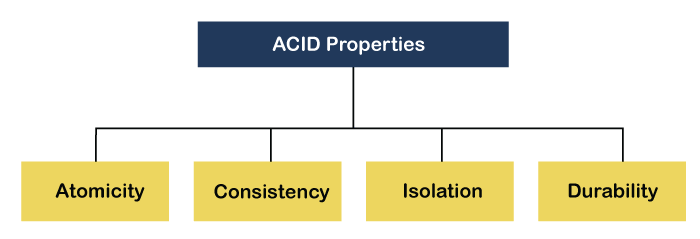
ACID Properties in DBMS

DBMS is the management of data that should remain integrated when any changes are done in it. It is because if the integrity of the data is affected, whole data will get disturbed and corrupted. Therefore, to maintain the integrity of the data, there are four properties described in the database management system, which are known as the **ACID** properties. The ACID properties are meant for the transaction that goes through a different group of tasks, and there we come to see the role of the ACID properties.

In this section, we will learn and understand about the ACID properties. We will learn what these properties stand for and what does each property is used for. We will also understand the ACID properties with the help of some examples.

ACID Properties

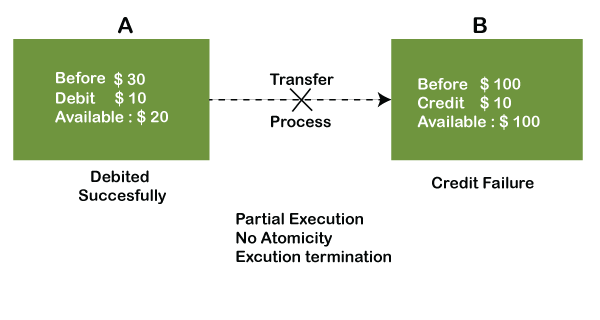
The expansion of the term ACID defines for:



1) Atomicity

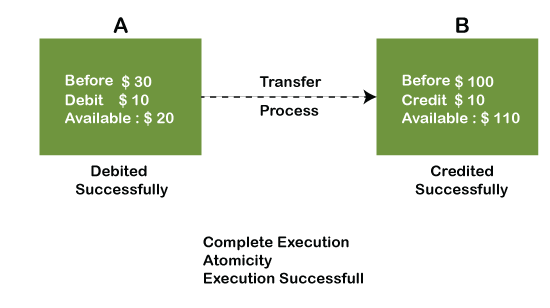
The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

**Example:** If Remo has account A having $30 in his account from which he wishes to send $10 to Sheero's account, which is B. In account B, a sum of $ 100 is already present. When $10 will be transferred to account B, the sum will become $110. Now, there will be two operations that will take place. One is the amount of $10 that Remo wants to transfer will be debited from his account A, and the same amount will get credited to account B, i.e., into Sheero's account. Now, what happens - the first operation of debit executes successfully, but the credit operation, however, fails. Thus, in Remo's account A, the value becomes $20, and to that of Sheero's account, it remains $100 as it was previously present.



In the above diagram, it can be seen that after crediting $10, the amount is still $100 in account B. So, it is not an atomic transaction.

The below image shows that both debit and credit operations are done successfully. Thus the transaction is atomic.

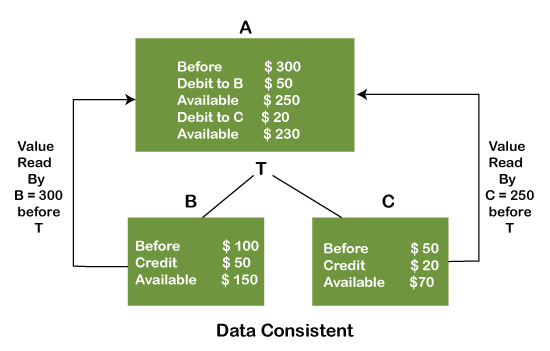


Thus, when the amount loses atomicity, then in the bank systems, this becomes a huge issue, and so the atomicity is the main focus in the bank systems.

2) Consistency

The word **consistency** means that the value should remain preserved always. In [DBMS](https://www.javatpoint.com/dbms-tutorial), the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

**Example:**

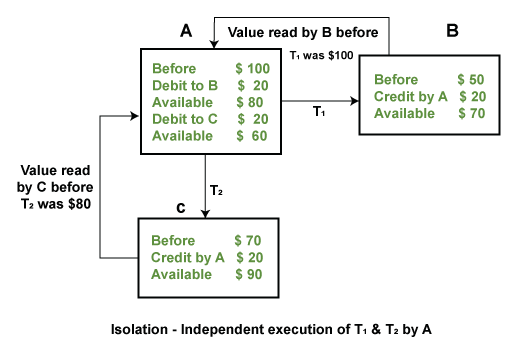


In the above figure, there are three accounts, A, B, and C, where A is making a transaction T one by one to both B & C. There are two operations that take place, i.e., Debit and Credit. Account A firstly debits $50 to account B, and the amount in account A is read $300 by B before the transaction. After the successful transaction T, the available amount in B becomes $150. Now, A debits $20 to account C, and that time, the value read by C is $250 (that is correct as a debit of $50 has been successfully done to B). The debit and credit operation from account A to C has been done successfully. We can see that the transaction is done successfully, and the value is also read correctly. Thus, the data is consistent. In case the value read by B and C is $300, which means that data is inconsistent because when the debit operation executes, it will not be consistent.

3) Isolation

The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

**Example:** If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.



4) Durability

Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes.

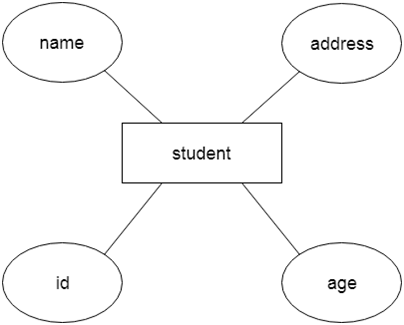
Therefore, the ACID property of DBMS plays a vital role in maintaining the consistency and availability of data in the database.

Thus, it was a precise introduction of ACID properties in DBMS. We have discussed these properties in the transaction section also.

# ER (Entity Relationship) Diagram in DBMS

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

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



Component of ER Diagra**m**



### **1. Entity:**

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

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



**a. Weak Entity**

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



### **2. Attribute**

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

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



**a. Key Attribute**

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



**b. Composite Attribute**

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



**c. Multivalued Attribute**

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

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

  
**d. Derived Attribute**

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

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



### **3. Relationship**

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



Types of relationship are as follows:

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

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

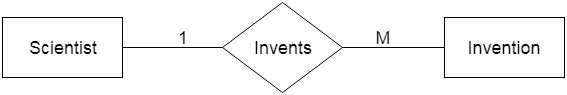
**For example,** A female can marry to one male, and a male can marry to one female.



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

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

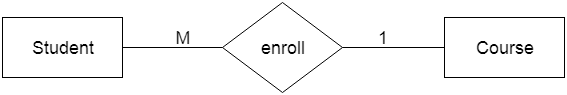
**For example,** Scientist can invent many inventions, but the invention is done by the only specific scientist.



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

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

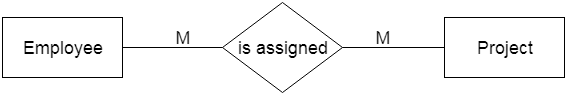
**For example,** Student enrolls for only one course, but a course can have many students.



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

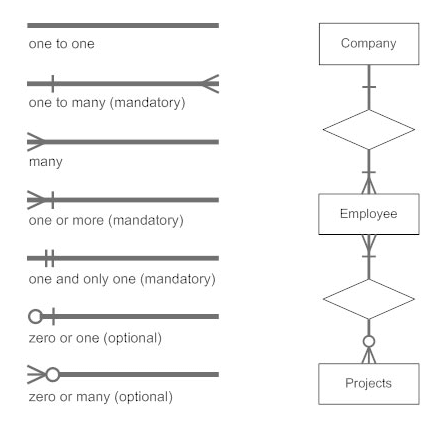
When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

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



Notation of ER diagram

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



**Fig: Notations of ER diagram**

# Cardinality in DBMS (Mapping Constraints)

## DBMS

DBMS stands for Database Management System, which is a tool, or a software used to do various operations on a Database like the Creation of the Database, Deletion of the Database, or Updating the current Database. To simplify processing and data querying, the most popular types of Databases currently in use typically model their data as rows and columns in a set of tables. The data may then be handled, updated, regulated, and structured with ease. For writing and querying data, most Databases employ Structured Query Language (SQL).

## Cardinality

Cardinality means how the entities are arranged to each other or what is the relationship structure between entities in a relationship set. In a Database Management System, Cardinality represents a number that denotes how many times an entity is participating with another entity in a relationship set. The Cardinality of DBMS is a very important attribute in representing the structure of a Database. In a table, the number of rows or tuples represents the Cardinality.

## Cardinality Ratio

Cardinality ratio is also called **Cardinality Mapping**, which represents the mapping of one entity set to another entity set in a relationship set. We generally take the example

of a binary relationship set where two entities are mapped to each other.

Cardinality is very important in the Database of various businesses. For example, if we want to track the purchase history of each customer then we can use the one-to-many cardinality to find the data of a specific customer. The Cardinality model can be used in Databases by Database Managers for a variety of purposes, but corporations often use it to evaluate customer or inventory data.

There are four types of Cardinality Mapping in Database Management Systems:

1. One to one
2. Many to one
3. One to many
4. Many to many

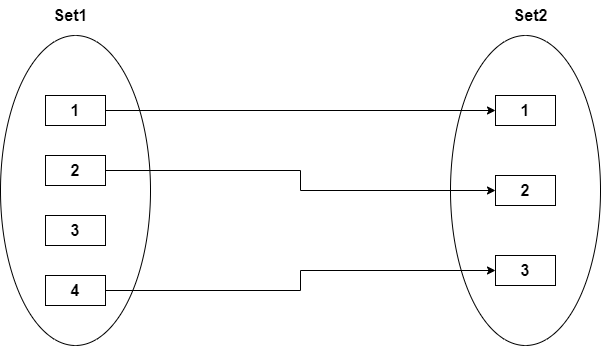
### **One to One**

One to one cardinality is represented by a **1:1** symbol. In this, there is at most one relationship from one entity to another entity. There are a lot of examples of one-to-one cardinality in real life databases.

**For example,** one student can have only one student id, and one student id can belong to only one student. So, the relationship mapping between student and student id will be one to one cardinality mapping.

Another example is the relationship between the director of the school and the school because one school can have a maximum of one director, and one director can belong to only one school.

#### **Note: it is not necessary that there would be a mapping for all entities in an entity set in one-to-one cardinality. Some entities cannot participate in the mapping.**

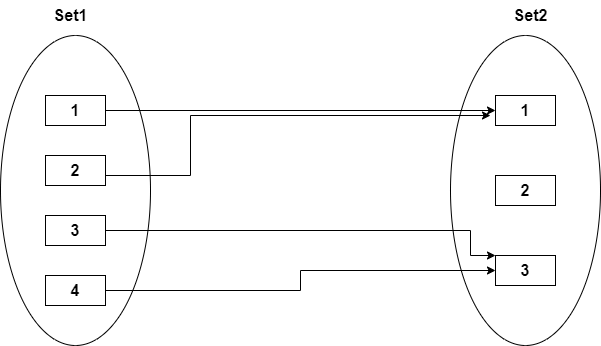


### **Many to One Cardinality:**

In many to one cardinality mapping, from set 1, there can be multiple sets that can make relationships with a single entity of set 2. Or we can also describe it as from set 2, and one entity can make a relationship with more than one entity of set 1.

One to one Cardinality is the subset of Many to one Cardinality. It can be represented by **M:1**.

**For example**, there are multiple patients in a hospital who are served by a single doctor, so the relationship between patients and doctors can be represented by Many to one Cardinality.

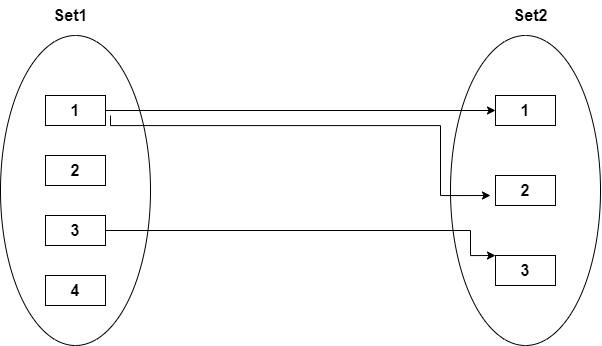


### **One to Many Cardinalities:**

In One-to-many cardinality mapping, from set 1, there can be a maximum single set that can make relationships with a single or more than one entity of set 2. Or we can also describe it as from set 2, more than one entity can make a relationship with only one entity of set 1.

One to one cardinality is the subset of One-to-many Cardinality. It can be represented by **1: M**.

**For Example,** in a hospital, there can be various compounders, so the relationship between the hospital and compounders can be mapped through One-to-many Cardinality.



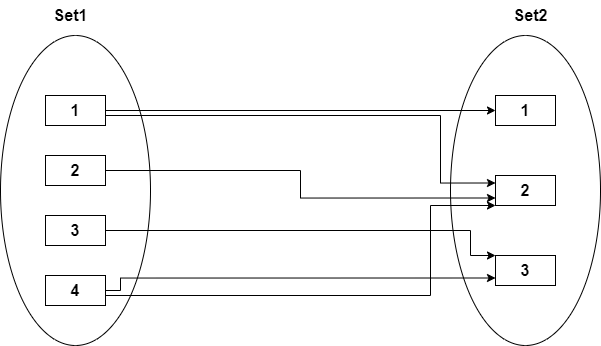
### **Many to Many Cardinalities:**

In many, many cardinalities mapping, there can be one or more than one entity that can associate with one or more than one entity of set 2. In the same way from the end of set 2, one or more than one entity can make a relation with one or more than one entity of set 1.

It is represented by **M: N** or **N: M**.

One to one cardinality, One to many cardinalities, and Many to one cardinality is the subset of the many to many cardinalities.

**For Example,** in a college, multiple students can work on a single project, and a single student can also work on multiple projects. So, the relationship between the project and the student can be represented by many to many cardinalities.



## Appropriate Mapping Cardinality

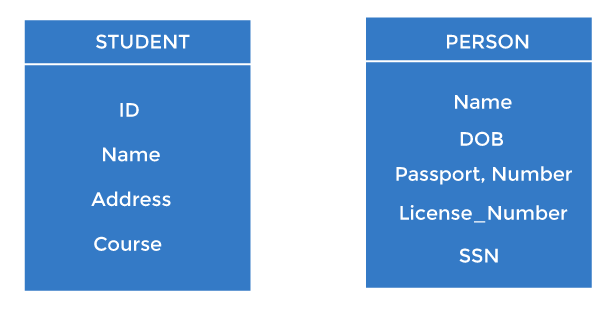
Evidently, the real-world context in which the relation set is modeled determines the Appropriate Mapping Cardinality for a specific relation set.

* We can combine relational tables with many involved tables if the Cardinality is one-to-many or many-to-one.
* One entity can be combined with a relation table if it has a one-to-one relationship and total participation, and two entities can be combined with their relation to form a single table if both of them have total participation.
* We cannot mix any two tables if the Cardinality is many-to-many.

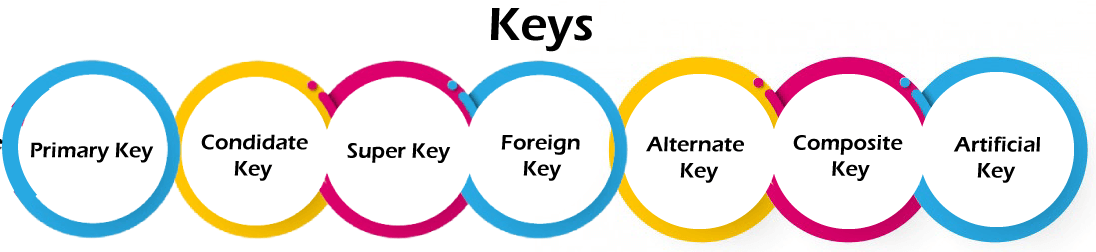
# Keys

* 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,** 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.

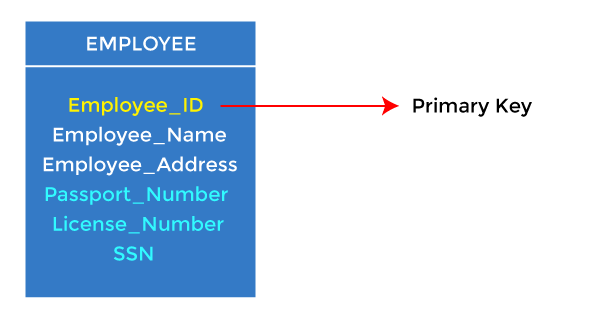


## Types of keys:



### **1. Primary key**

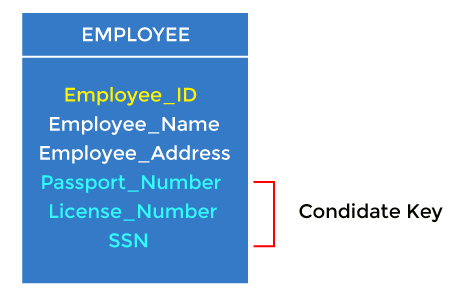
* It is the first key used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.
* In the EMPLOYEE table, ID can be the primary key since it is unique for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



### **2. Candidate key**

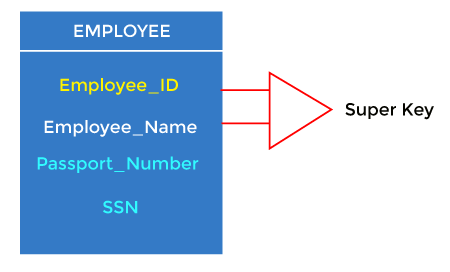
* A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
* Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

**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.

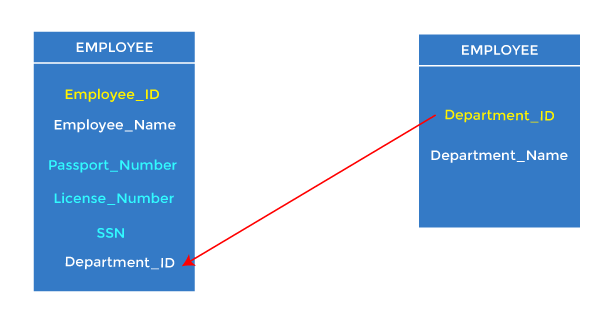


**For example:** In the above EMPLOYEE table, for(EMPLOEE\_ID, EMPLOYEE\_NAME), the name of two employees can be the same, but their EMPLYEE\_ID can't be the same. Hence, this combination can also be a key.

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

### **4. Foreign key**

* Foreign keys are the column of the table 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 two different entities. So we can't store the department's information 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.
* In the EMPLOYEE table, Department\_Id is the foreign key, and both the tables are related.



### **5. Alternate key**

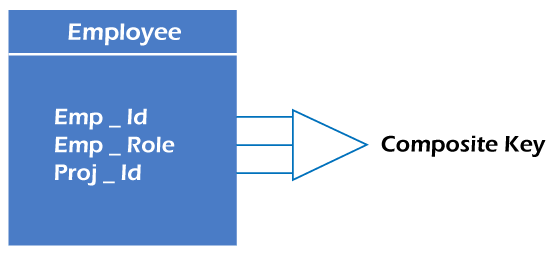
There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternate key. **In other words,** the total number of the alternate keys is the total number of candidate keys minus the primary key. The alternate key may or may not exist. If there is only one candidate key in a relation, it does not have an alternate key.

**For example,** employee relation has two attributes, Employee\_Id and PAN\_No, that act as candidate keys. In this relation, Employee\_Id is chosen as the primary key, so the other candidate key, PAN\_No, acts as the Alternate key.

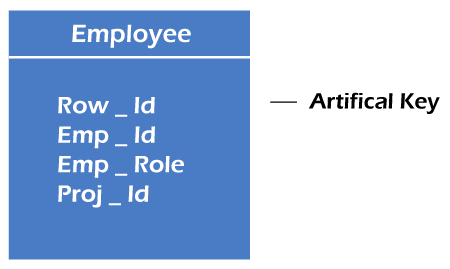


### **6. Composite key**

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



**For example,** in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously. So the primary key will be composed of all three attributes, namely Emp\_ID, Emp\_role, and Proj\_ID in combination. So these attributes act as a composite key 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 a primary key is large and complex and has no relationship with many other relations. The data values of the artificial keys are usually numbered in a serial order.

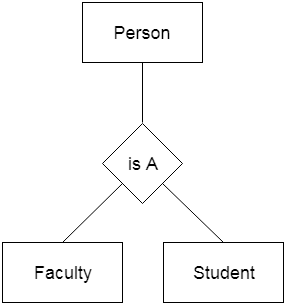
**For example,** the primary key, which is composed of Emp\_ID, Emp\_role, and Proj\_ID, is large in employee relations. So it would be better to add a new virtual attribute to identify each tuple in the relation uniquely.

**ADDITIONAL FEATURES OF ER MODEL**

Generalization

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

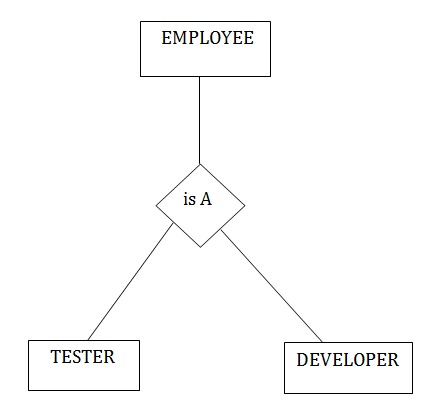
**For example,** Faculty and Student entities can be generalized and create a higher level entity Person.



Specialization

* Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.
* Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.
* Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

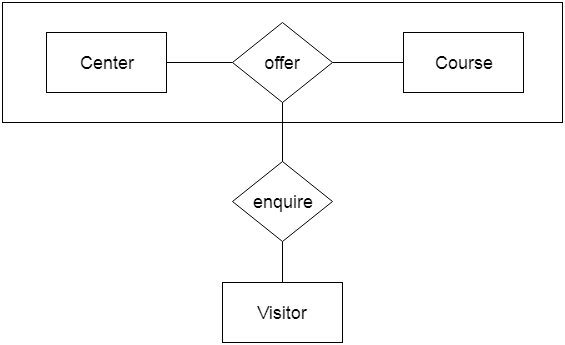
**For example:** In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.



# Aggregation

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

**For example:** Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.



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

**Introduction to the Relational Model in DBMS**

The **Relational Model(RM)** for database management is an approach to logically represent and manage the data stored in a database. In this model, the data is organized into a collection of **two-dimensional inter-related** tables, also known as **relations**. Each relation is a collection of columns and rows, where the column represents the attributes of an entity and the rows (or tuples) represents the records.

**Terminologies used in Realation Model**

* **Tables:** relations are saved in the table format. A table has two properties rows and columns
* **Attribute:** columns represent as attributes
* **Tuple:** A Row represent as Tuple
* **Relation Schema:** A relation schema represents the name of the relation with its attributes.
* **Degree:** The total number of attributes which in the relation is called the degree of the relation.
* **Cardinality:** Total number of rows present in the Table.
* **Column:** The column represents the set of values for a specific attribute.
* **Relation instance:** The set of tuples of a relation at a particular instance of time is called as relation instance.

**Advantages of Realation Model**

* **Structural Independence**: Structural independence is an ability that allows us to make changes in one database structure without affecting other. The relational model have structural independence. Hence making required changes in the database is convenient in relational database model.
* **Conceptual Simplicity**: The relational model allows the designer to simply focus on logical design and not on physical design. Hence relational models are conceptually simple to understand.
* **Query Capability**: Using simple query language (such as SQL) user can get information from the database or designer can manipulate the database structure.
* **Easy design,maintenance and usage**: The relational models can be designed logically hence they are easy to maintain and use.

**Disadvantages of Realation Model**

* Relational model requires powerful hardware and large data storage devices.
* May lead to slower processing time.
* Poorly designed systems lead to poor implementation of database systems.

**Integrity constraint over relations in DBMS**

Integrity constraints are rules that help to maintain the accuracy and consistency of data in a database. For example, a simple integrity constraint in DBMS might state that all students must have a valid Roll Number. This would prevent someone from accidentally entering an invalid roll number into the database.

Integrity constraints can also be used to enforce relationships between tables. For example, if a student can only have one aadhaar number, then an integrity constraint can be used to ensure that only one aadhaar number is entered for each student.

**Different types of Integrity Constraints**

* **Domain Constraint**
* **Entity Integer Constraint**
* **Referential Integrity Constraint**
* **Key Constraints**

**Domain Constraint**

A domain constraint is a restriction on the values that can be stored in a column. Strings, character, time, integer, currency, date etc. Are examples of the data type of domain constraints.

example, if you have a column for "age" domain integrity constraints in DBMS would ensure that only integer values can be entered into that column. This ensures that only valid data is entered into the database.

**Entity Integer Constraint**

Entity integrity constraints would ensure that null values are not entered into any required columns. It states that primary key value can't be null. This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows

For example, if you have a column for "roll\_number" an entity integrity constraint in DBMS would ensure that this column cannot contain any null values.

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*PRIMARY KEY(column)*

*);*

**Referential Integrity Constraint**

A referential integrity constraint is a restriction on how foreign keys can be used. A foreign key is a column in one table that references a primary key in another table.

For example, let's say you have a table of Students and a table of Marks. The "roll\_number" column in the Marks table would be a foreign key that references the "roll\_number" column in the Students table.

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*FOREIGN KEY (column) REFERENCES table\_name1(column)*

*);*

**Key Constraints**

A key constraint is a rule that defines how data in a column(s) can be stored in a table. A key is composed of one or more columns whose values uniquely identify each row in the table. There are several different types of key constraints in DBMS, each with its own specific purpose.

**1. Unique Key**

A unique key refers to a column or a set of columns that identify every record uniquely in a table. All the values in this key would have to be unique. values of a unique key won’t allow duplicate values and it is only capable of having one null value.

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*UNIQUE (column)*

*);*

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*CONSTRAINT constraint\_name UNIQUE(column)*

*);*

**2. Primary Key**

The primary key refers to a column of a table that helps us identify all the records uniquely present in that table. Any table can consist of only a single primary key constraint. values of a primary key won’t allow null value or a duplicate values.

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*CONSTRAINT constraint\_name PRIMARY KEY(column)*

*);*

**3. Foreign Key**

We use a foreign key to establish relationships between two available tables. The foreign key would require every value present in a column/set of columns to match the referential table’s primary key. A foreign key helps us to maintain data as well as referential integrity.

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*CONSTRAINT constraint\_name FOREIGN KEY (column)*

*REFERENCES table\_name1(column)*

*);*

**4. Composite Key**

The composite key refers to a set of multiple attributes that help us uniquely identify every tuple present in a table. The attributes present in a set may not be unique whenever we consider them separately. Thus, when we take them all together, it will ensure total uniqueness.

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*CONSTRAINT constraint\_name UNIQUE(column,column)*

*);*

**5. Super Key**

A super key refers to the set of all those keys that help us uniquely identify all the rows present in a table. It means that all of these columns present in a table that can identify the columns of that table uniquely act as the super keys.

**6. Candidate Key**

The candidate keys refer to those attributes that identify rows uniquely in a table. In a table, we select the primary key from a candidate key. Thus, a candidate key has similar properties as that of the primary keys that we have explained above. In a table, there can be multiple candidate keys.

**7. Alternate Key**

As we have stated above, any table can consist of multiple choices for the primary key. But, it can only choose one. Thus, all those keys that did not become a primary key are known as alternate keys.

**CHECK Constraint**

The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a column it will allow only certain values for this column. If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*CONSTRAINT constraint\_name CHECK (condition)*

*);*

**Example:**

CREATE TABLE students (

ID int,

LastName varchar(30),

FirstName varchar(30),

Age int,

CONSTRAINT con\_age CHECK (Age>=18)

);

**DEFAULT Constraint**

The DEFAULT constraint is used to set a default value for a column. The default value will be added to all new records, if no other value is specified.

The DEFAULT constraint can also be used to insert system date, by using functions like GETDATE().

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype DEFAULT 'value',*

*column3 datatype DEFAULT GETDATE(),*

*....*

*);*

**Example:**

CREATE TABLE students (

ID int,

LastName varchar(30),

FirstName varchar(30),

Age int DEFAULT 18,

joining\_date DEFAULT GETDATE()

);

**Enforcing Integrity Constraints in DBMS**

Integrity Constraints are specified when a relation is created and enforced when a relation is modified. The impact of domain, PRIMARY KEY, and UNIQUE constraints is straightforward: If an insert, delete, or update command causes a violation, it is rejected. Every potential Integrity violation is generally checked at the end of each SQL statement execution, although it can be deferred until the end of the transaction executing the statement.

On the other hand, insertions of Students tuples do not violate referential integrity, and deletions of Students tuples could cause violations.

SQL provides several alternative ways to handle foreign key violations. We must consider three basic questions:

***1. What should we do if an Enrolled row is inserted, with a student\_id column value that does not appear in any row of the Students table?***

In this case, the INSERT command is simply rejected.

***2. What should we do if a Students row is deleted?***

The options are:

☑      Delete all Enrolled rows that refer to the deleted Students row.

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*FOREIGN KEY (column) REFERENCES table\_name1(column) ON DELETE CASCADE*

*);*

☑      Disallow the deletion of the Students row if an Enrolled row refers to it.

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*FOREIGN KEY (column) REFERENCES table\_name1(column) ON DELETE NO ACTION*

*);*

☑      Set the studid column to the sid of some (existing) 'default' student, for every Enrolled row that refers to the deleted Students row.

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*FOREIGN KEY (column) REFERENCES table\_name1(column) ON DELETE SET DEFAULT*

*);*

***3. What should we do if the primary key value of a Students row is updated?***

The options here are similar to the previous case. Replace *ON DELETE with ON UPDATE*

*Syntax in Mysql*

*CREATE TABLE table\_name (*

*column1 datatype,*

*column2 datatype,*

*column3 datatype,*

*....*

*FOREIGN KEY (column) REFERENCES table\_name1(column) ON UPDATE NO ACTION*

*);*

**Example:**

CREATE TABLE Enrolled (

Sid int,

Cid varchar(30),

joining\_date DEFAULT GETDATE()

FOREIGN KEY (Sid) REFERENCES Students

ON DELETE CASCADE

ON UPDATE NO ACTION

);

**Querying Relational Data in DBMS**

A relational database query is a question about the data, and the answer consists of a new relation containing the result. For example, we might want to find all students AGE less than 18 or all students enrolled in perticular course.

The ***SELECT*** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

*Syntax in Mysql*

*SELECT column1, column2, ...*

*FROM table\_name;*

If you want to select all the fields available in the table, use the following syntax:

*Syntax in Mysql*

*SELECT \* FROM table\_name;*

The symbol ´**\***´ means that we retain all fields of selected tuples in the result.

We can retrieve rows corresponding to students who are younger than 18 withthe following SQL query:

**Example:**

SELECT \* FROM Students WHERE age < 18;

The condition **age < 18** in the WHERE clause specifies that we want to select only tuples in which the age field has a value less than 18.

In addition to selecting a subset of tuples, a query can extract a subset of the fields of each selected tuple. we can compute the student\_id and First\_name of students who are younger than 18 with the following query:

**Example:**

SELECT ID,FirstName FROM Students WHERE age < 18;

**SQL Aliases**

Aliases are the temporary names given to tables or columns. An alias is created with the **AS** keyword.

*Alias Column Syntax in Mysql*

*SELECT column\_name AS alias\_name*

*FROM table\_name;*

*Alias Table Syntax in Mysql*

*SELECT column\_name(s)*

*FROM table\_name AS alias\_name;*

**Example:**

SELECT studentID AS ID,

FROM students AS S;

**Aliases can be useful when:**

* There are more than one table involved in a query
* Functions are used in the query
* Column names are big or not very readable
* Two or more columns are combined togeth

**SELECT data from Multiple Tables**

We can also combine information from multiple tables.

*Syntax in Mysql*

*SELECT table1.column1, table2.column2*

*FROM table1, table2*

*WHERE table1.column1 = table2.column1;*

**Example:**

SELECT S.name, E.cid

FROM Students AS S, Enrolled AS E

WHERE S.sid = E.sid;

**Views in DBMS**

A view is a table whose rows are not explicitly stored, a view is a virtual table based on the result-set of an SQL statement. A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

A view is generated to show the information that the end-user requests the data according to specified needs rather than complete information of the table.

**Advantages of View over database tables**

* Using Views, we can join multiple tables into a single virtual table.
* Views hide data complexity.
* In the database, views take less space than tables for storing data because the database contains only the view definition.
* Views indicate the subset of that data, which is contained in the tables of the database.

**Creating Views**

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables or another view.  
To create a view, a user must have the appropriate system privilege according to the specific implementation.

*Syntax in Mysql*

*CREATE VIEW view\_name AS*

*SELECT column1, column2, ...*

*FROM table\_name*

*WHERE condition;*

**Example:**

CREATE VIEW Students\_CSE AS

SELECT Roll\_no,Name

FROM Students

WHERE Branch = 'CSE';

**Updating a View**

A view can be updated with the CREATE OR REPLACE VIEW statement.

*Syntax in Mysql*

*CREATE OR REPLACE VIEW view\_name AS*

*SELECT column1, column2, ...*

*FROM table\_name*

*WHERE condition;*

The following SQL adds the "Mobile" column to the "Students\_CSE" view:

**Example:**

CREATE OR REPLACE VIEW Students\_CSE AS

SELECT Roll\_no,Name,Mobile

FROM Students

WHERE Branch = 'CSE';

**CREATE VIEW** defines a view on a set of tables or views or both. **REPLACE VIEW** redefines an existing view or, if the specified view does not exist,

**Inserting a row in a view**

We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.

*Syntax in Mysql*

*INSERT INTO view\_name(column1, column2, ...)*

*VALUES(value1,value2,.....);*

**Example:**

INSERT INTO Students\_CSE(Roll\_no,Name,Mobile)

VALUES(521,'ram',9988776655);

**Deleting a row in a view**

Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view.

*Syntax in Mysql*

*DELETE FROM view\_name*

*WHERE condition;*

**Example:**

DELETE FROM Students\_CSE

WHERE Name="ram";

**Querying a View**

We can query the view as follows

*Syntax in Mysql*

*SELECT \* FROM view\_name*

**Example:**

SELECT \* FROM Students\_CSE;

**Dropping a View**

In order to delete a view in a database, we can use the DROP VIEW statement.

*Syntax in Mysql*

*DROP FROM view\_name*

**Example:**

DROP FROM Students\_CSE;

**Relational Algebra in DBMS**

Relational Algebra is not actually implemented anywhere. It is collection of mathematical expressions. So it is a theoretical or logical model to access the database.

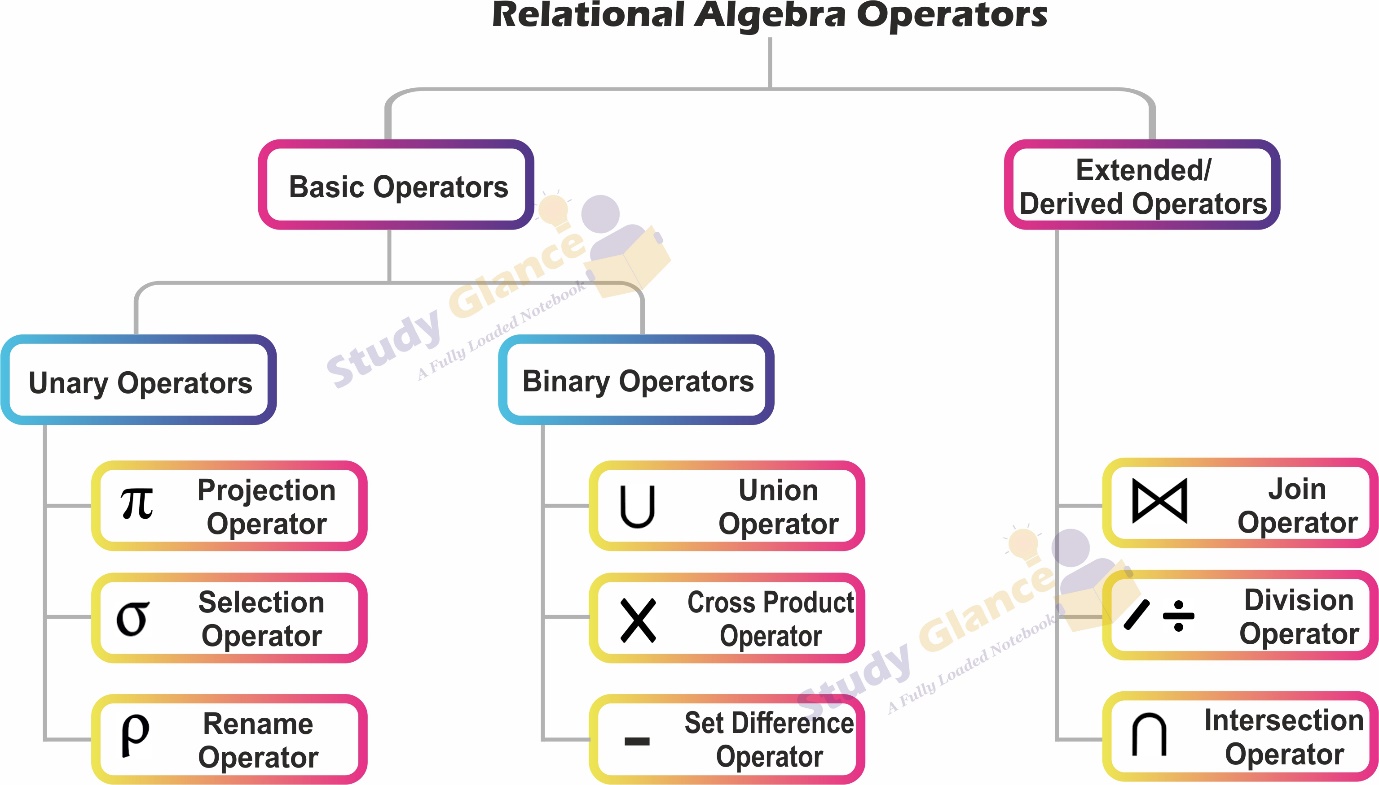
In simple words we can say, Relation Algebra provides the base to SQL and Non-SQL. SQL and Non-SQL are physically implemented.

Relational Algebra is a procedural query language or formal query language, which takes instances of one or more relation as an input and generates a new relation as an output. It uses a different set of operators (like unary or binary operators) and operands to perform queries.

In procedural query language every user has to mention two things to access data

**What to do?  
How to do?**

The primary operations of relational algebra has classified as



**Projection Operator(ℼ)**

Projection Operator (**ℼ**) displays the columns of a single relation(Table) R based on the specified attributes and eliminating duplicates.

*Syntax*

∏attribute list(R)∏attribute list(�)

*Example: Retrieve the Roll\_no and name of the students*

∏Roll\_no,Name(Students)∏Roll\_no,Name(Students)

*Example: Retrieve the name of the students*

∏Name(Students)∏Name(Students)

Note: This operator removes the redundancy

**Selection Operator(σ)**

Selection Operator (**σ**) performs a selection operation. It selects complete rows or tuples from the table which satisfies the selection condition.

*Syntax*

σselection\_condition(R)�selection\_condition(�)

*Example:*

σBranch='cse'(Students)�Branch='cse'(Students)

**Important Points**

1) We may use logical operators like ∧ , ∨ , ! and relational operators like = , ≠ , > , < , <= , >= with the selection condition.

2) Selection operator only selects the required tuples according to the selection condition. It does not display the selected tuples. To display the selected tuples, projection operator is used.

3) Selection operator always selects the entire tuple. It can not select a section or part of a tuple.

4) Selection operator is commutative in nature i.e.

σA∧B(R)=σB∧A(R)�A∧B(�)=�B∧A(�)

5) The number of rows returned by a selection operation is obviously less than or equal to the number of rows in the original table

.

* Minimum Cardinality = 0
* Maximum Cardinality = |R|

*Example: Retrieve the name of the students whose age is < 18;*

∏Name(σage<18(R))∏Name(�age<18(�))

**Rename Operator**

Rename operation is denoted by "Rho"(**ρ**). As its name suggests it is used to rename the resultant relation.

*Syntax*

Px(R)��(�)

where the symbol ‘ρ’ is used to denote the RENAME operator and R is the result which is saved with the name X.

*Example: Retrieve Roll\_no, Name of the students who admited in CSE and rename to cse\_students*

Pcse\_students(∏Roll\_no,Name(σBranch='cse'(Students)))�cse\_students(∏Roll\_no,Name(�Branch='cse'(Students)))

**Union Operator**

Union operator is denoted by **∪** symbol and it is used to select all the tuples from two relations(R1 and R2).

*Syntax*

R1 ∪ R2R1 ∪ R2

To perform union operation, R1 and R2 must be the same number of attributes with same order and same domains. In the result, Duplicate tuples will automatically removed

**Cartesian/CROSS Product**

Cartesian or CROSS Product denoted by **X**. It is used to combine each row in one Relation(R1) with each row in the other Relation(R2).

*Syntax*

R1 X R2R1 X R2

**Difference Operator**

Difference Operator denoted by intersection minus (**-**). The difference between two relations(R1 and R2) is a relation which includes all tuples that are in R1 but not in R2.

*Syntax*

R1 − R2R1 − R2

**Intersection operator**

Intersection operator is denoted by **∩** symbol and it is used to select common tuples from two relations(R1 and R2)

*Syntax*

R1 ∩ R2R1 ∩ R2

To perform Intersection operation, R1 and R2 must be the same number of attributes with same order and same domains. In the result, Duplicate tuples will automatically removed

**Division Operator**

Division Operator denoted by **/ or ÷**. We can apply Division operator on two Relations(R1 and R2) if and only if: Attributes of R2 is proper subset of Attributes of R1 and when query contain **all or every**.

*Syntax*

R1 ÷ R2R1 ÷ R2

**Join Operations**

Join operator denoted by **⨝**. A Join operation combines related tuples from different relations.

Join operators can apply, if and only if there must be some common attributes in two relations, It is used when we need to access data from more than one Relations.

**Join operator = cross product + condition**

Types of JOINs:

**1. Inner Join**

     Natural Join

     Equi Join

     Theta Join

**2. Outter Join**

     Left Outter Join

     Right Outter Join

     Full Outter Join

**1. Inner Join**

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let’s study various types of Inner Joins

**Natural Join**

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same(one as a primary key and other as foreign key).

*Syntax*

R1 ⨝ R2R1 ⨝ R2

**Equi Join**

In Equi join we uses only equivalence condition while performing join between two relations.

*Syntax*

R1 ⨝(... = ...)R2R1 ⨝(... = ...)R2

**Theta Join**

Theta Join is used to join two relations based on some conditions(θ). The condition can be on any attributes of the relations performing Theta join. Any comparison operator can be used in the condition.

*Syntax*

R1 ⨝θR2R1 ⨝�R2

**2. Outter Join**

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

**Left Outter Join (⟕)**

*Syntax*

R1 ⟕ R2R1 ⟕ R2

**Right Outter Join (⟖)**

*Syntax*

R1 ⟖ R2R1 ⟖ R2

**Full Outter Join (⟗)**

*Syntax*

R1 ⟗ R2R1 ⟗ R2

**Tuple Relational Calculus in DBMS**  
  
Relational calculus is a Declarative Langauge or non-procedural query language.

Before understanding Relational calculus in DBMS, we need to understand Procedural Language and Declarative Langauge.

**Procedural Language** - Those Languages which clearly define how to get the required results from the Database are called Procedural Language. Relational algebra is a Procedural Language.

**Declarative Language** - Those Language that only cares about What to get from the database without getting into how to get the results are called Declarative Language. Relational Calculus is a Declarative Language.

Relational Calculus in database management system (DBMS) is all about **"What you want ?"**. Relational calculus does not tell us how to get the results from the Database, but it just cares about what we want.

**Types of Relational Calculus in DBMS**

1. **Tuple Relational Calculus (TRC)**
2. **Domain Relational Calculus (DRC)**

**Tuple Relational Calculus (TRC)**

Tuple Relational Calculus uses a tuple variable (t) that goes to each row of the table and checks if the predicate condition is true or false for the given row. Depending on the given predicate condition, it returns the row or part of the row.

A tuple relational calculus query has the form

*Syntax*

*{t | P(t)}*

where t is a tuple variable and P(t) is a logical formula that describes the conditions that the tuples in the result must satisfy. The curly braces { } are used to indicate that the expression is a set of tuples.

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.

**There are two types of tuple variables**

**Free Variables:** Any tuple variable without any ‘For All’ or ‘there exists’ condition is called Free Variable  
**Bounded Variables:** Any tuple variable with the ‘For All’ or ‘there exists’ condition is called Bounded Variable

*Example: Retrieve the name of the students whose age is < 18*

T.name | Student(T) AND T.age < 18

**Domain Relational Calculus (DRC)**

Domain Relational Calculus uses domain Variables to get the column values required from the database based on the predicate expression or condition.

The key idea of Domain Relational Calculus is to define queries in terms of the domains (sets of values) of the attributes, rather than the tuples or records. Queries are formulated as expressions involving variables that range over domains.

A Domain Relational Calculus query has the form

*Syntax*

*{<x1,x2,x3,x4...> | P(x1,x2,x3,x4...)}*

where each Xi is either a domain variable or a constant, p(x1, x2, ... ,xn ) denotes a DRC formula composed of atoms.

*Example: Retrieve the name and branch of the students whose age is < 18*

{< name, branch > | ∈ Student ∧ age < 18}