Database system applications

Database systems have a wide range of applications across various industries and domains. Here are some common applications:

1. **Enterprise Resource Planning (ERP) Systems**: ERP systems like SAP, Oracle E-Business Suite, and Microsoft Dynamics utilize databases to manage various business processes including finance, human resources, supply chain management, and customer relationship management.
2. **Customer Relationship Management (CRM)**: CRM systems such as Salesforce, HubSpot, and Zoho CRM store and manage customer data, interactions, and sales-related information.
3. **E-commerce Platforms**: Platforms like Amazon, eBay, and Shopify rely on databases to store product information, customer data, orders, and transactions.
4. **Content Management Systems (CMS)**: CMS platforms like WordPress, Drupal, and Joomla use databases to store website content, user information, and configuration settings.
5. **Social Media Platforms**: Social media platforms like Facebook, Twitter, and Instagram use databases to store user profiles, posts, comments, and interactions.
6. **Banking and Finance**: Banking systems utilize databases for managing customer accounts, transactions, loans, and financial records.
7. **Healthcare Systems**: Electronic Health Record (EHR) systems and hospital management systems use databases to store patient information, medical records, treatment history, and billing data.

database systems versus file systems

| **Feature** | **Database Systems** | **File Systems** |
| --- | --- | --- |
| Data Structure | Structured data organized in tables and relationships. | Unstructured or semi-structured data stored in files. |
| Data Integrity | Support for enforcing data integrity constraints. | Limited support for data integrity. |
| Query Language | SQL (Structured Query Language) for querying data. | No standardized query language (depends on file format). |
| Scalability | Typically scalable to handle large volumes of data. | Limited scalability for handling large datasets. |
| Concurrency | Support for concurrent access and transactions. | Limited support for concurrent access and transactions. |
| Data Redundancy | Minimizes data redundancy through normalization. | Data redundancy depends on how files are organized. |
| ACID Properties | Transactions adhere to ACID (Atomicity, Consistency, Isolation, Durability). | ACID properties may not be guaranteed. |
| Data Relationships | Support for defining and maintaining relationships between data entities. | No built-in support for managing data relationships. |
| Performance | Optimized for fast data retrieval and manipulation. | Performance may vary depending on file access patterns. |
| Data Security | Granular access control and encryption capabilities. | Limited security features, often relies on file system permissions. |
| Complexity | Higher complexity due to relational model and features. | Lower complexity, easier to understand and implement. |

**Views of Data**

Views of data refer to the different ways to show the database management system to the user hiding its complexity.

Levels of abstraction:

The database is highly complex data. Developers hide the complexity of data structure which might not be familiar with the end-users. So, they provide abstract views of data. This is called data abstraction. There are 3 levels of abstraction. They are:

* Physical level
* Logical Level
* View Level

Physical Level:

It is called the lowest level of abstraction or internal level. It describes how the data is actually stored in the database. At this level, complex data structure details are shown.

Logical Level:

It is called the middle level of abstraction or conceptual level. It describes what data is stored in a database and also shows what relationship exists among those data.

View Level:

It is called the highest level of abstraction or external level. It describes the user interaction with database systems via application programs, that hide details of data types. It shows only a part of the entire database.

**Schemas and instances:**

Instances are defined as the data stored in a database at a particular moment of time. Schemas are the overall design of a database. The instances are actually copies of the instant data and information. But, on the other hand, schemas don’t contain any data or any information.

**Data independence:**

Data independence refers to the mechanism of keeping the data independent from the user. Actually, meaning it allows users to make any kind of changes from one level of schemas without hampering other users at different levels of schemas.

**3 level architecture:**

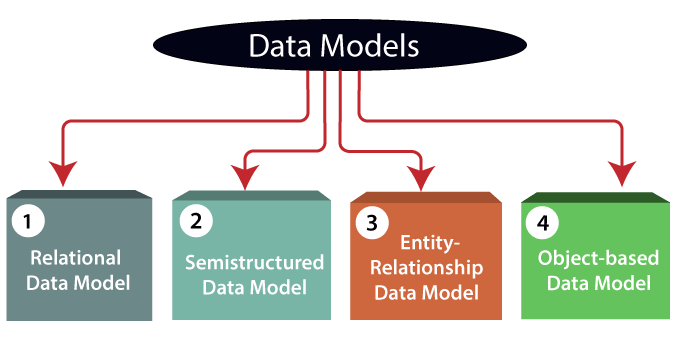
* View Level
* Logical Level
* Physical Level

These three architecture makes the data independent from the users. Users can easily access data. However, the database is also convenient for the users from the view level, as they can get a simple user interface. Secondly, the logical level or conceptual level makes sure that if any ‘A’ user changes the structure of the table, then the other user cannot find the changes done by ‘A’. Thirdly, the physical level controls the view from users so they cannot view how data are actually stored in the database.

data models

Data Models

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



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

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

**3) Object-based Data Model:** An extension of the ER model with notions of functions, encapsulation, and object identity, as well. This model supports a rich type system that includes structured and collection types. Thus, in 1980s, various database systems following the object-oriented approach were developed. Here, the objects are nothing but the data carrying its properties. Skip 10s

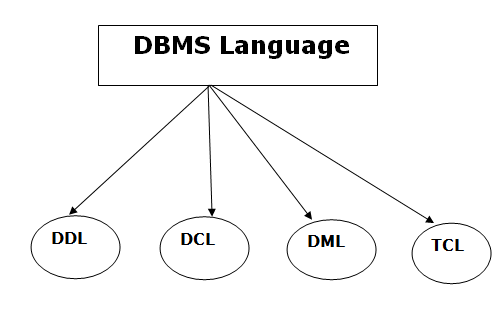
**4) Semistructured Data Model:** This type of data model is different from the other three data models (explained above). The semistructured data model allows the data specifications at places where the individual data items of the same type may have different attributes sets. The Extensible Markup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

database languages

# 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**



## **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.

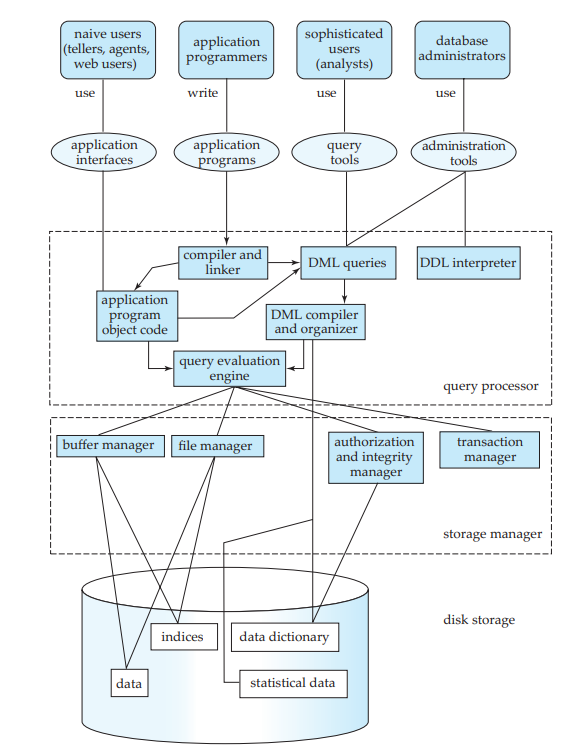
transaction management

Transaction management in Database Management Systems (DBMS) ensures that database operations are performed reliably, consistently, and concurrently. Transactions are logical units of work that consist of one or more database operations, such as insertion, deletion, or modification of data. Here's an overview of transaction management in DBMS:

1. **ACID Properties**:
   * **Atomicity**: Ensures that either all operations in a transaction are completed successfully, or none of them are. If any operation fails, the entire transaction is rolled back, and the database is restored to its original state.
   * **Consistency**: Guarantees that the database remains in a consistent state before and after the transaction. Each transaction must adhere to all integrity constraints, such as foreign key constraints and uniqueness constraints.
   * **Isolation**: Ensures that transactions operate independently of each other, as if they were executed serially. Transactions are isolated from each other to prevent interference and ensure data integrity.
   * **Durability**: Ensures that the changes made by committed transactions persist even in the event of system failures. Committed data must be stored permanently and remain unaffected by crashes or restarts.
2. **Transaction States**:
   * **Active**: The initial state of a transaction where it is actively executing operations.
   * **Partially Committed**: The state reached when all operations in the transaction have been completed successfully, but the changes have not been made permanent yet.
   * **Committed**: The state reached when the changes made by the transaction have been successfully written to the database and are now permanent.
   * **Failed**: The state reached when an error occurs during the execution of the transaction, causing it to be terminated and rolled back.
   * **Aborted**: The state reached when a transaction that has failed is rolled back to its original state.
3. **Concurrency Control**:
   * Concurrency control mechanisms prevent interference between concurrent transactions to maintain data consistency and integrity. Techniques such as locking, timestamp ordering, and multi-version concurrency control (MVCC) are used to manage concurrent access to the database.
   * Lock-based concurrency control involves acquiring locks on data items to ensure exclusive access by one transaction at a time.
   * MVCC allows transactions to operate on a snapshot of the database, ensuring that each transaction sees a consistent view of the data, regardless of concurrent modifications by other transactions.
4. **Transaction Management Commands**:
   * **BEGIN TRANSACTION**: Marks the beginning of a transaction.
   * **COMMIT**: Marks the successful completion of a transaction, making its changes permanent.
   * **ROLLBACK**: Aborts the transaction and rolls back any changes made by it, restoring the database to its original state.
   * **SAVEPOINT**: Sets a named point within a transaction to which you can later roll back if necessary.

Transaction management is a critical component of DBMS, ensuring data integrity, consistency, and reliability in multi-user and concurrent environments. Proper transaction management ensures that databases maintain their integrity and reliability even in the face of failures and concurrent access by multiple users or applications.

database system structure



**Th­ree Parts that make up the Database System are:**

* Query Processor
* Storage Manager
* Disk Storage

**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](https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/) statements. It is responsible for updating, storing, deleting, and retrieving data in the database.   
It contains the following components –

* **Authorization Manager:**It ensures role-based access control, i.e,. checks whether the particular person is privileged to perform the requested operation or not.
* **Integrity Manager:**It checks the integrity constraints when the database is modified.
* **Transaction Manager:**It controls concurrent access by performing the operations in a scheduled way that it receives the transaction. Thus, it ensures that the database remains in the consistent state before and after the execution of a transaction.
* **File Manager:**It manages the file space and the data structure used to represent information in the database.
* **Buffer Manager:**It is responsible for cache memory and the transfer of data between the secondary storage and main memory.

**3. Disk Storage:** It contains the following components –

* **Data Files:**It stores the data.
* **Data Dictionary:**It contains the information about the structure of any database object. It stores the collection of names, definitions, and attributes for data elements that are being used in a database. The Data Dictionary stores metadata, i.e., data about the database.
* **Indices:**It provides faster retrieval of data item.

application architecture

Data modeling using the Entity Relationship

# Introduction of ER Model

The Entity Relational Model is a model for identifying entities to be represented in the database and representation of how those entities are related. The ER data model specifies enterprise schema that represents the overall logical structure of a database graphically.

The Entity Relationship Diagram explains the relationship among the entities present in the database. ER models are used to model real-world objects like a person, a car, or a company and the relation between these real-world objects. In short, the 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 into 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 these symbols:

* **Rectangles:**Rectangles represent Entities in the ER Model.
* **Ellipses:**Ellipses represent Attributes in the 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](https://iotap.geeksforgeeks.org/problems/what-is-the-difference-between-single-valued-and-multi-valued-attributes).
* **Double Rectangle:**Double Rectangle represents a Weak Entity.

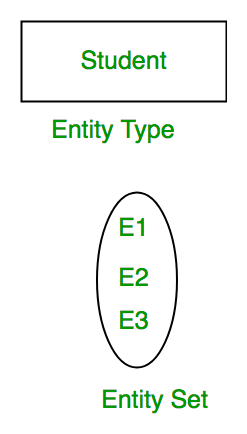
## Components of ER Diagram

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

### Entity

An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may 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, E1 is an entity having Entity Type Student and the set of all students is called Entity Set. In ER diagram, Entity Type is represented as:



*Entity Set*

#### 1. Strong Entity

A [Strong Entity](https://www.geeksforgeeks.org/difference-between-strong-and-weak-entity/) is a type of entity that has a key Attribute. Strong Entity does not depend on other Entity in the Schema. It has a primary key, that helps in identifying it uniquely, and it is represented by a rectangle. These 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](https://www.geeksforgeeks.org/weak-entity-set-in-er-diagrams/).

**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**.

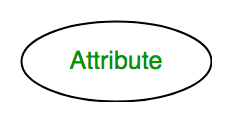
A weak entity type is represented by a Double Rectangle. The participation of weak entity types is always total. The relationship between the weak entity type and its identifying strong entity type is called identifying relationship and it is represented by a double diamond.



*Strong Entity and Weak Entity*

### ****Attributes****

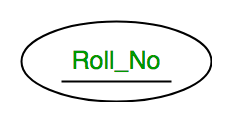
[Attributes](https://www.geeksforgeeks.org/types-of-attributes-in-er-model/) are the properties that define the entity type. For example, Roll\_No, Name, DOB, Age, Address, and Mobile\_No are the attributes that define entity type Student. In ER diagram, the attribute is represented by an oval.



*Attribute*

#### **1. Key Attribute**

The attribute which **uniquely identifies each entity** in the entity set is called the key attribute. For example, Roll\_No will be unique for each student. In ER diagram, the key attribute is represented by an oval with 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**

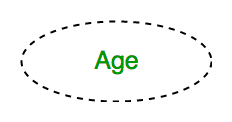
An attribute consisting of more than one value for a given entity. For example, Phone\_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval.



*Multivalued Attribute*

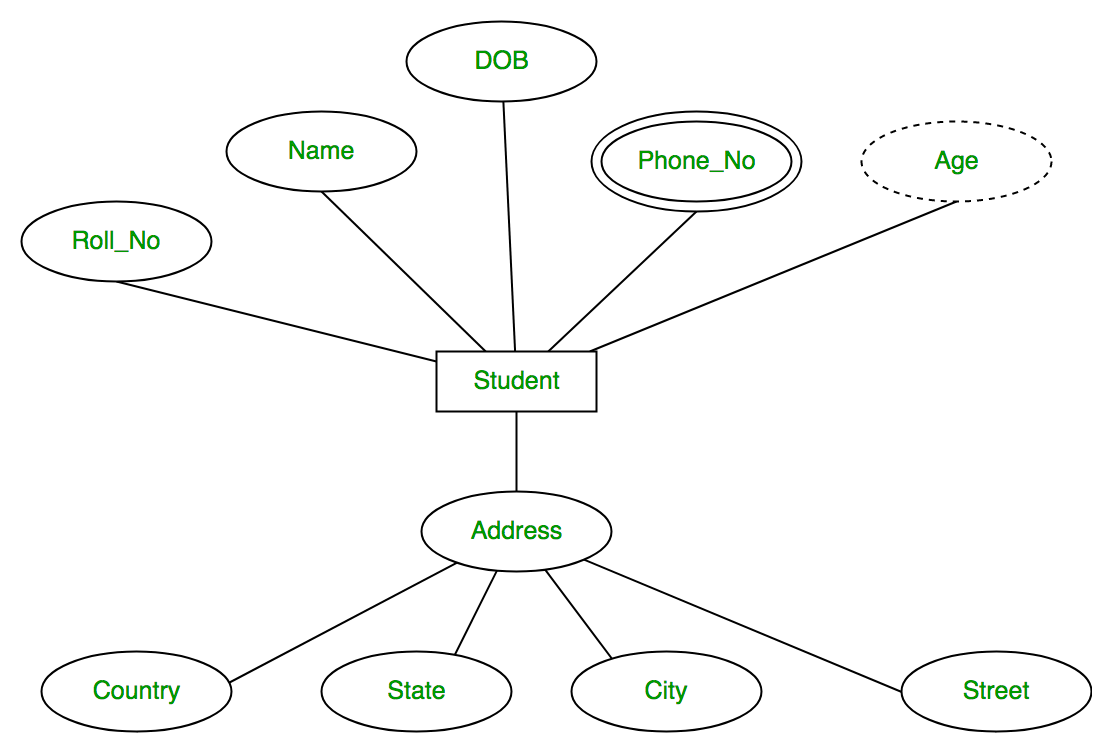
#### **4. Derived Attribute**

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*

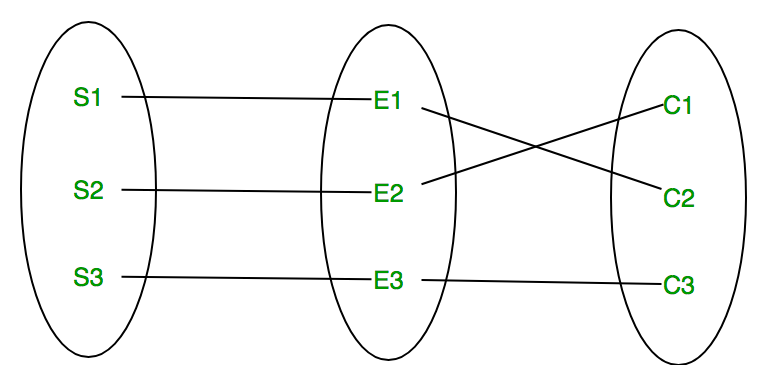
### ****Relationship Type and Relationship Set****

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.

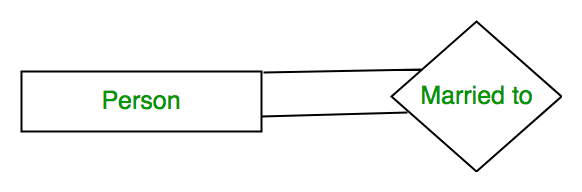


*Relationship Set*

#### **Degree of a Relationship Set**

The number of different entity sets participating in a relationship set is called the [degree of a relationship set.](https://www.geeksforgeeks.org/degree-of-relations-in-dbms/)

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



*Unary Relationship*

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



*Binary Relationship*

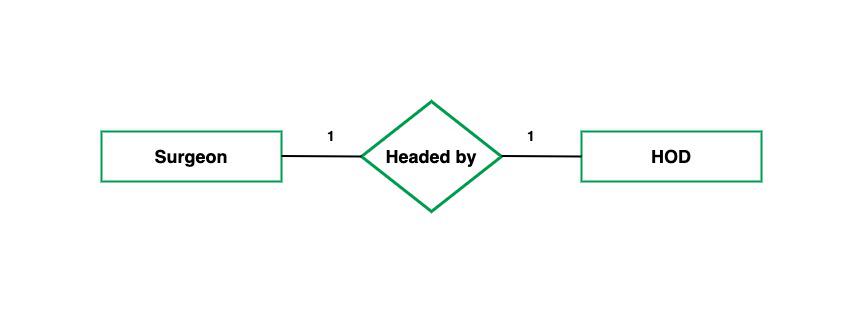
**3. n-ary Relationship:**When there are n entities set participating in a relation, the relationship is called an n-ary relationship.

#### **Cardinality**

The number of times an entity of an entity set participates in a relationship set is known as [cardinality](https://www.geeksforgeeks.org/cardinality-in-dbms/). Cardinality can be of different types:

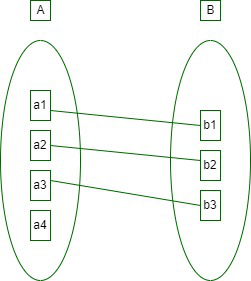
**1. One-to-One:** When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one. Let us assume that a male can marry one female and a female can marry one male. So the relationship will be one-to-one.

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



*one to one cardinality*

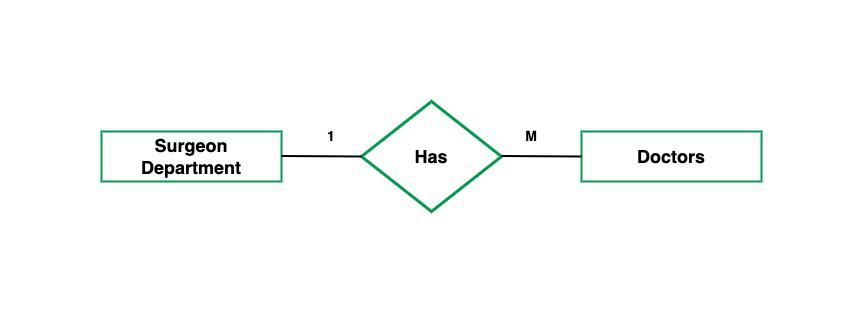
Using Sets, it can be represented as:



*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. Let us assume that one surgeon deparment can accomodate many doctors. So the Cardinality will be 1 to M. It means one deparment has many Doctors.

total number of tables that can used is 3.



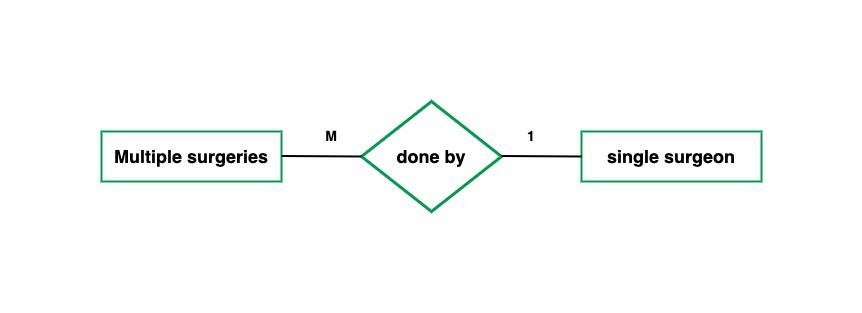
*one to many cardinality*

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

*Set Representation of One-to-Many*

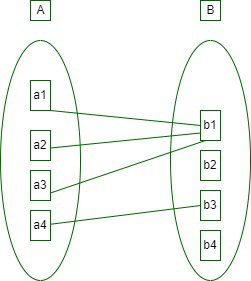
**3. Many-to-One:** When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by 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 cardinality*

Using Sets, it can be represented as:

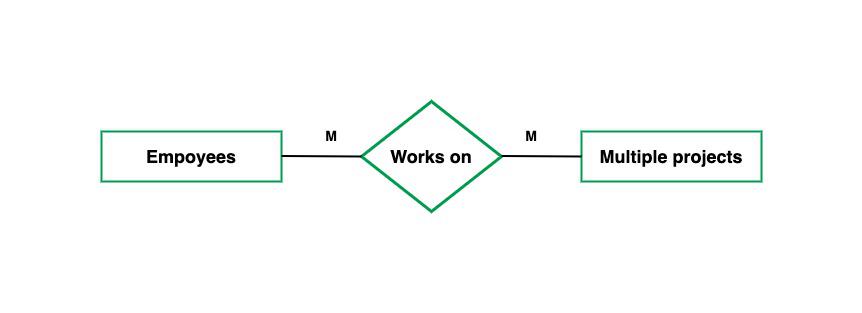


*Set Representation of Many-to-One*

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

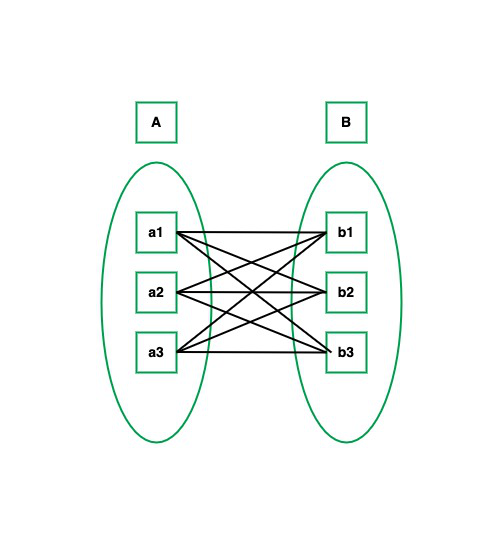
**4. Many-to-Many:**When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

the total number of tables that can be used in this is 3.



*many to many cardinality*

Using Sets, it can be represented as:



*Many-to-Many Set Representation*

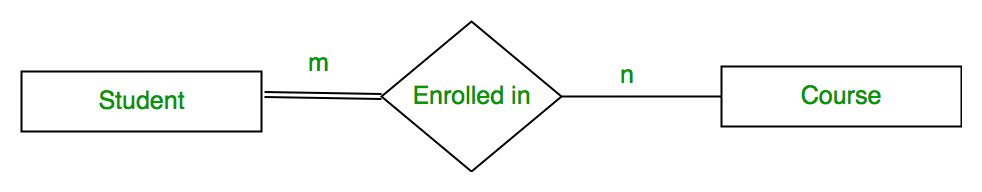
In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3, and S4. So it is many-to-many relationships.

### ****Participation Constraint****

[Participation Constraint](https://www.geeksforgeeks.org/structural-constraints-of-relationships-in-er-model/) is applied to the entity participating in the relationship set.

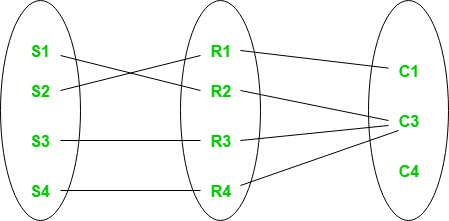
**1. Total Participation –** Each entity in the entity setmust participate in the relationship. If each student must enroll in a course, the participation of students will be total. Total participation is shown by a double line in the ER diagram.

**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.

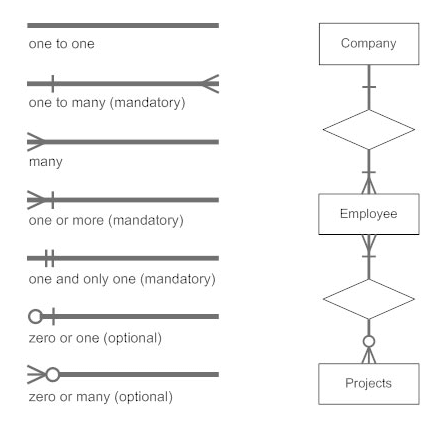


*Total Participation and Partial Participation*

Using Set, it can be represented as,



Model: ER model concepts, notation for ER diagram



mapping constraints

one to many and many to one

concepts of super key

# Super Key in DBMS

We can define a super key as a set of those keys that identify a row or a tuple uniquely. The word super denotes the superiority of a key. Thus, a super key is the superset of a key known as a **Candidate key** (discussed in the next section). It means a [candidate key](https://www.javatpoint.com/candidate-key-in-dbms) is obtained from a super key only.

Here, we will discuss about the super key, i.e., what is the role of a super key, how to use it, and will be looking at some practical examples that will help us to understand it in a better way.

## **Role of Super Key**

The role of the super key is simply to identify the tuples of the specified table in the database. It is the superset where the candidate key is a part of the super key only. So, all those attributes in a table that is capable of identifying the other attributes of the table in a unique manner are all super keys.

#### **Note: Two or more attributes in a table can together identify a table uniquely, so the combination of such attributes is nothing but a super key only.**

## **Examples of Super Key**

Let's consider an **EMPLOYEE\_DETAIL** table example where we have the following attribute:

Backward Skip 10sPlay VideoForward Skip 10s

ADVERTISEMENT

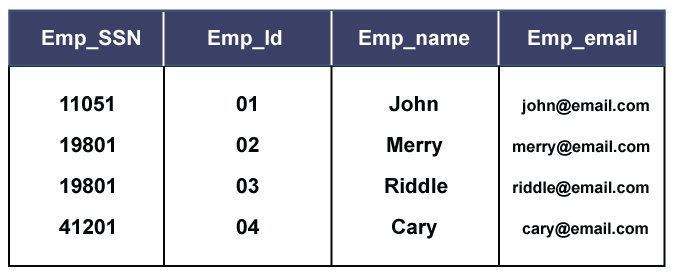
**Emp\_SSN:** The SSN number is stored in this field.

**Emp\_Id:** An attribute that stores the value of the employee identification number.

**Emp\_name:** An attribute that stores the name of the employee holding the specified employee id.

**Emp\_email**: An attribute that stores the email id of the specified employees.

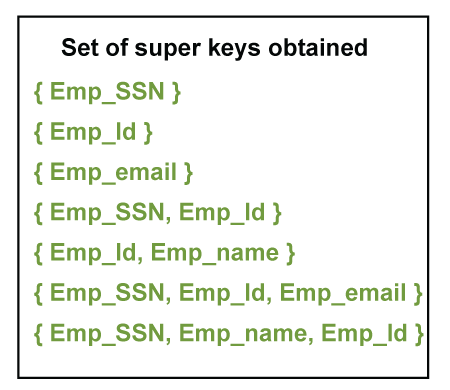
The **EMPLOYEE\_DETAIL** table is given below that will help you understand better:



So, from the above table, we conclude the following set of the super keys:

ADVERTISEMENT

ADVERTISEMENT



These all are the set of super keys which, together or combining with other prime attributes, can identify a table uniquely.

Just like, if we set Super key on Emp\_SSN, it will be able to identify all other tuples of the table very easily. Similarly, if we set the Super key on (Emp\_Id, Emp\_name}, we can easily get the value or details of the other remaining attributes of the employee. So, in this way, we can create and search out the super keys from a table.

candidate key

# Candidate Key in DBMS

A **candidate key** is a part of a key known as **Super Key** (discussed in the previous section), where the super key is the super set of all those attributes that can uniquely identify a table.

Here, we will be discussing about candidate key, its role, as well as its use. We will also look at some examples that will make us to better understand the concept of a candidate key.

## **What is a Candidate Key**

A candidate key is a subset of a super key set where the key which contains no redundant attribute is none other than a **Candidate Key**. In order to select the candidate keys from the set of super key, we need to look at the super key set.

## **Role of a Candidate Key**

The role of a candidate key is to identify a table row or column uniquely. Also, the value of a candidate key cannot be Null. The description of a candidate key is "no redundant attributes" and being a "minimal representation of a tuple," according to the Experts.

ADVERTISEMENT

### How a Candidate key is different from a Primary Key

Although the purpose of both candidate and the primary key is the same, that is to uniquely identify the tuples, and then also they are different from each other. It is because, in a table, we can have one or more than one candidate key, but we can create only one primary key for a table. Thus, from the number of obtained candidate keys, we can identify the appropriate primary key. However, if there is only one candidate key in a table, then it can be considered for both key constraints.

## **Example of Candidate Key**

Let's look at the same example took while discussing Super Key to understand the working of a candidate key.

We have an **EMPLOYEE\_DETAIL** table where we have the following attributes:

**Emp\_SSN:** The SSN number is stored in this field.

**Emp\_Id:** An attribute that stores the value of the employee identification number.

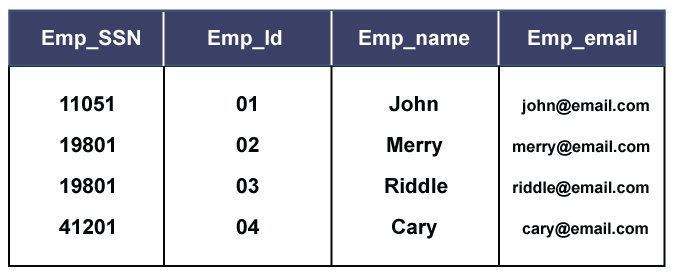
**Emp\_name:** An attribute that stores the name of the employee holding the specified employee id.

ADVERTISEMENT

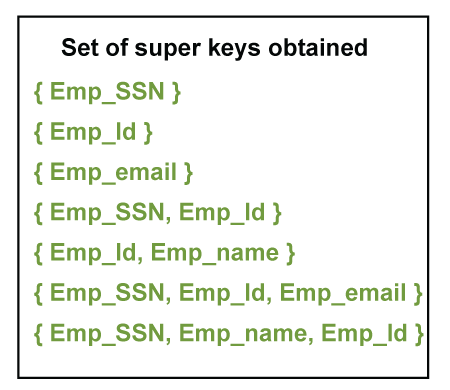
ADVERTISEMENT

**Emp\_email:** An attribute that stores the email id of the specified employees.

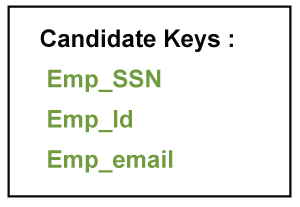
The **EMPLOYEE\_DETAIL** table is given below that will help you understand better:



So, from the above table, we obtained the below given super keys (discussed in the previous section):



Now, from these sets of super keys, we can conclude the candidate keys. In order to pick up the candidate keys, the best way is to analyze and form the primary keys as much as we can. So, we need to identify those sets from the super key sets that alone can identify the whole table, or we can say the other attributes of the table. Thus, the result is:



So, these are the three attributes obtained that can identify the other non-prime attributes of the table. All these are the candidate keys and from which we can pick the most appropriate attribute that can easily identify all records of the table, which will be described as the Primary key.

## **Difference between Candidate Key and Super Key**

From the above discussions, we can have the following difference points:

|  |  |
| --- | --- |
| **Super Key** | **Candidate Key** |
| It is the superset of all such attributes that can uniquely identify the table. | It is the subset or the part of the Super key. |
| It is not at all compulsory that all super keys are candidate keys. | On the other hand, all candidate keys are super keys. |
| The super key attribute can be NULL, which means its values can be null. | An attribute holding a candidate key can never be NULL, which means its values cannot be null. |
| All the super keys formed together to bring the candidate keys. | Similarly, candidate keys are put together to create primary keys. |
| The number of super keys formed is always seen more. | Here, Candidate keys are less than super keys. |

primary key

# Primary Key in DBMS

There are certain keys in DBMS that are used for different purposes, from which the most commonly known is the Primary Key.

Here, in this section, we will look at the Primary key - What it is, what is the use of a primary key, and we will also implement some examples to understand that how a primary key works.

## **What is a Primary Key**

A Primary Key is the minimal set of attributes of a table that has the task to uniquely identify the rows, or we can say the tuples of the given particular table.

A primary key of a relation is one of the possible candidate keys which the database designer thinks it's primary. It may be selected for convenience, performance and many other reasons. The choice of the possible primary key from the candidate keys depend upon the following conditions.

* **Minimal:** The primary key should be composed of the minimum number of attributes that satisfy unique occurrences of the tuples. So if one candidate key is formed using two attributes and another using a single attribute then the one with the single attribute key should be chosen as the primary key.
* **Accessible:** The primary key used should be accessible by anyone who wants to use it. The user must easily insert, access or delete a tuple using it.
* **NON NULL Value:** The primary key must have a non-null value for each tuple of the relation, which is required for the identification of the tuple.
* **Time Variant:** The values of the primary key must not change or become null during the time of a relation.
* **Unique:** The value of the primary key must not be duplicated in any of the tuples of a relation.

### Syntax for creating primary key constraint:

**The primary key constraint can be defined at the column level or table level.**

**At column level:**

1. **<column\_name><datatype>** Primary key;

**At table level:**

1. Primary key(**<column\_name1>**[,column\_name**>**]....);

### Properties of a Primary Key:

* A relation can contain only one primary key.
* A primary key may be composed of a single attribute known as single primary key or more than one attribute known as composite key.
* A primary key is the minimum super key.
* The data values for the primary key attribute should not be null.
* Attributes which are part of a primary key are known as Prime attributes.
* Primary key is always chosen from the possible candidate keys.
* If the primary key is made of more than one attribute then those attributes are irreducible.
* We use the convention that the attribute that form primary key of relation is underlined.
* Primary key cannot contain duplicate values.
* Columns that are defined as LONG or LONG RAW cannot be part of a primary key.

## **Use of Primary Key**

As defined above, a primary key is used to uniquely identify the rows of a table. Thus, a row that needs to be uniquely identified, the key constraint is set as the Primary key to that particular field. A primary key can never have a **NULL** value because the use of the primary key is to identify a value uniquely, but if no value will be there, how could it sustain. Thus, the field set with the primary key constraint cannot be NULL. Also, it all depends on the user that the user can add or delete the key if applied.

#### **Note: One table can have only one primary key where it does not matter the table having one or more columns in it.**

## **Understanding Primary Key**

Let's discover some examples through which we can understand the role and use of a Primary key. Generally, in a database, we apply the primary key on those tuples or columns through which we need to uniquely identify the other database fields.

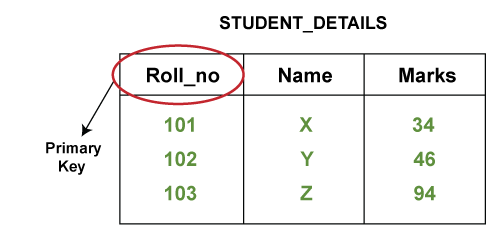
**For example:** When we store the registration details of the students in the database, we find the registration number field unique and assign the primary key to the field. Also, for an employee table, we set the primary key on the employee Id of the table.

ADVERTISEMENT

ADVERTISEMENT

Let's understand it practically:

Below is the table named **STUDENT\_DETAILS**, where Roll\_no, Name, and Marks are the specified attributes of it.



As we know that from these three attributes, the Roll\_no attribute is the one that can uniquely identify other two attributes of the table as each student is provided with a unique roll number in every organization. So, we can set the primary key constraint on the Roll\_no column.

**What if we set Name as Primary Key?**

If we set the primary key on the Name attribute, it will be invalid because two or more students can have the same name. So, if we set the primary key on Name and try to enter the same name for two students, it will display an error. That's why we cannot set the Name attribute as the primary key.

**What if we set Marks as Primary Key?**

If we set the primary key on the Marks attribute, then it will be an inappropriate approach because two or more students can score similar marks in a subject. Thus, if we set the Marks attribute as the primary key, we will not be able to enter the same score for other entity. That's why we cannot set the primary key for the Marks attribute.

We can see the given below table to understand it:

### Miscellaneous Example

Consider another example where we have a table named **PRODUCT\_DETAILS**where we have the following attributes:

**COMP\_Id:** The company id from where we may purchase different products.

ADVERTISEMENT

**Prod\_Name:** It intakes the products that we purchased.

**Prod\_Id:** Each product is given an identification number.

ADVERTISEMENT

ADVERTISEMENT

**Points to be noted:**

* We cannot set the primary key for **Prod\_Name** because we can purchase the same product from two different companies.
* On the other hand, both **COMP\_Id** and **Prod\_Id** can uniquely identify a company and a product. So, the question is on which attribute we should set the primary key, or we should set the primary key to both attributes, i.e., Prod\_Id and Comp\_Id.
* If we set the primary key on COMP\_Id, then it will be wrong because we can purchase more than one type of product from the same company. So, if we will set the primary key for COMP\_Id, then we will be unable to use that company for purchasing more different products.
* Also, we cannot set the primary key for both Prod\_Id and COMP\_Id as there can be only one primary key in one table.
* If we set the primary key for Prod\_Id, the approach will be correct because each product we purchase will have a unique identification. So, we should set the primary key for **Prod\_Id**.

We can understand it better by looking at the below table:



## **Defining the Primary Key**

Let's discuss how we can set a primary key on a STUDENT\_DETAILS table attribute:

### Creating a Primary Key

Below is the syntax for creating Primary Key on Roll\_no attribute of STUDENT\_DETAILS table:

1. CREATE TABLE STUDENT\_DETAIL (
2. Roll\_no **int** NOT NULL PRIMARY KEY,
3. Name varchar (200) NOT NULL,
4. Marks **int** NOT NULL
5. } ;

It is the basic syntax used in SQL and Oracle servers.

### Removing Primary Key

It is also possible to delete the set primary key from an attribute using **ALTER** and **DROP** commands.

1. ALTER TABLE STUDENT\_DETAIL DROP PRIMARY KEY ;

### Adding Primary Key after creating the table

In order to set the primary key after creating a table, use the **ALTER** command and add the primary key constraint to do so. The syntax is shown below:

1. ALTER TABLE STUDENT\_DETAIL
2. ADD CONSTRAINT PK\_STUDENT\_DETAIL PRIMARY KEY (Roll\_no, Name);

We have taken the Name attribute just for understanding the syntax.

So, in this way, we can use and set the primary key on a table. However, the syntax for defining the primary key may vary for different types of databases.

unique key

Unique Key in SQL

A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.

You can say that it is little like primary key but it can accept only one null value and it cannot have duplicate values.

The unique key and primary key both provide a guarantee for uniqueness for a column or a set of columns.

There is an automatically defined unique key constraint within a primary key constraint.

Backward Skip 10sPlay VideoForward Skip 10s

There may be many unique key constraints for one table, but only one PRIMARY KEY constraint for one table.

**SQL UNIQUE KEY constraint on CREATE TABLE:**

If you want to create a UNIQUE constraint on the "S\_Id" column when the "students" table is created, use the following SQL syntax:

**SQL Server / Oracle / MS Access:**

**(Defining a unique key constraint on single column):**

1. **CREATE** **TABLE** students
2. (
3. S\_Id **int** NOT NULL **UNIQUE**,
4. LastName **varchar** (255) NOT NULL,
5. FirstName **varchar** (255),
6. City **varchar** (255)
7. )

**MySQL:**

1. **CREATE** **TABLE** students
2. **CREATE** **TABLE** students
3. (
4. S\_Id **int** NOT NULL,
5. LastName **varchar** (255) NOT NULL,
6. FirstName **varchar** (255),
7. City **varchar** (255),
8. **UNIQUE** (S\_Id)
9. )

**(Defining a unique key constraint on multiple columns):**

**MySQL / SQL Server / Oracle / MS Access:**

1. **CREATE** **TABLE** students
2. (
3. S\_Id **int** NOT NULL,
4. LastName **varchar** (255) NOT NULL,
5. FirstName **varchar** (255),
6. City **varchar** (255),
7. **CONSTRAINT** uc\_studentId **UNIQUE** (S\_Id, LastName)
8. )

**SQL UNIQUE KEY constraint on ALTER TABLE:**

If you want to create a unique constraint on "S\_Id" column when the table is already created, you should use the following SQL syntax:

**(Defining a unique key constraint on single column):**

**MySQL / SQL Server / Oracle / MS Access:**

1. **ALTER** **TABLE** students
2. **ADD** **UNIQUE** (S\_Id)

**(Defining a unique key constraint on multiple columns):**

ADVERTISEMENT

**MySQL / SQL Server / Oracle / MS Access:**

1. **ALTER** **TABLE** students
2. **ADD** **CONSTRAINT** uc\_StudentId **UNIQUE**  (S\_Id, LastName)

**DROP SYNTAX FOR A FOREIGN KEY constraint:**

If you want to drop a UNIQUE constraint, use the following SQL syntax:

**MySQL:**

1. **ALTER** **TABLE** students
2. **DROP** **INDEX** uc\_studentID

**SQL Server / Oracle / MS Access:**

1. **ALTER** **TABLE** students
2. **DROP** **CONSTRAINT** uc\_studentID

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.

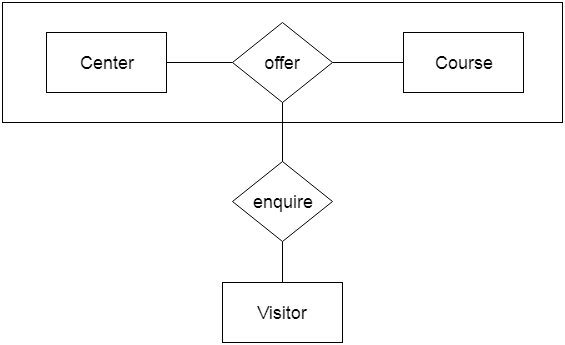
**For example,** Faculty and Student entities can be generalized and create a higher level entity Person.

aggregation

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.

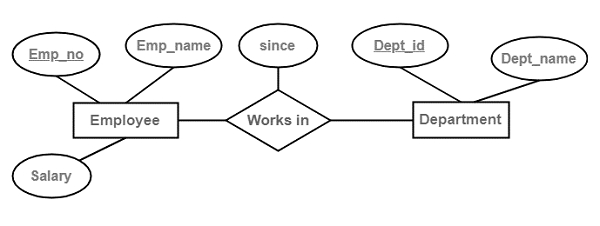
**For example:** Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will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



reduction of an ER diagram to tables.

Entity relationship diagram is the graphical representation of entities and relationships among those entities in the database.

**Example**



## **Conversion of ER diagrams to tables**

Follow the steps given below for the conversion of the ER diagrams to tables in the database management system (DBMS) −

**Step 1** − Conversion of strong entities

* For each strong entity create a separate table with the same name.
* Includes all attributes, if there is any composite attribute divided into simple attributes and has to be included.
* Ignore multivalued attributes at this stage.
* Select the p key for the table.

**Step 2** − Conversion of weak entity

* For each weak entity create a separate table with the same name.
* Include all attributes.
* Include the P key of a strong entity as foreign key is the weak entity.
* Declare the combination of foreign key and decimator attribute as P key from the weak entity.

**Step 3** − Conversion of one-to-one relationship

* For each one to one relation, say A and B modify either A side or B side to include the P key of the other side as a foreign key.
* If A or B is having total participation, then that should be a modified table.
* If a relationship consists of attributes, include them also in the modified table.

**Step 4** − Conversion of one-to-many relationship

* For each one to many relationships, modify the M side to include the P key of one side as a foreign key.
* If relationships consist of attributes, include them as well.

**Step 5** − Conversion of many-many relationship

* For each many-many relationship, create a separate table including the P key of M side and N side as foreign keys in the new table.
* Declare the combination of foreign keys as P for the new table.
* If relationships consist of attributes, include them also in the new table.

**Step 6** − Conversion of multivalued attributes

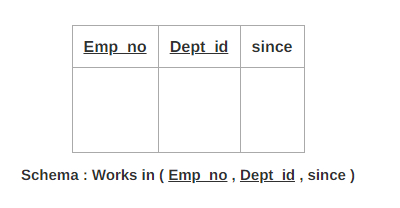
* For each multivalued attribute create a separate table and include the P key of the present table as foreign key.
* Declare the combination of foreign key and multivalued attribute as P keys.

**Step 7** − Conversion of n-ary relationship

* For each n-ary relationship create a separate table and include the P key of all entities as foreign key.
* Declare the combination of foreign keys as P key.

## **Table**

After successful conversion, the result will be as follows −



UNIT-2

Relational model: Structure of relational databases, relational algebra, tuple relational calculus, domain relational calculus. SQL: Characteristics of SQL, advantages of SQL, types of SQL commands, SQL operators and their procedure, tables, views and indexes, queries and sub-queries, aggregate functions, insert, update and delete operations, joins, union, intersection, minus, cursors in SQL. Domain constraints, referential integrity, assertions, triggers, authorization and authentication. Relational database design & normalization: Functional dependencies, normal forms- First, second, third, BCNF, fourth and fifth normal forms, decomposition

Relational model: Structure of relational databases

E.F. Codd proposed the relational Model to model data in the form of relations or tables. After designing the conceptual model of the Database using [ER diagram](https://www.geeksforgeeks.org/introduction-of-er-model/), we need to convert the conceptual model into a relational model which can be implemented using any [RDBMS](https://www.geeksforgeeks.org/difference-between-rdbms-and-dbms/) language like Oracle SQL, MySQL, etc. So we will see what the Relational Model is.

The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as relations. The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type. The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.

## **What is the Relational Model?**

The relational model represents how data is stored in Relational Databases. A relational database consists of a collection of tables, each of which is assigned a unique name. Consider a relation STUDENT with attributes ROLL\_NO, NAME, ADDRESS, PHONE, and AGE shown in the table.   
  
**Table Student**

| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| --- | --- | --- | --- | --- |
| 1 | RAM | DELHI | 9455123451 | 18 |
| 2 | RAMESH | GURGAON | 9652431543 | 18 |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |
| 4 | SURESH | DELHI |  | 18 |

## **Important Terminologies**

* **Attribute:** Attributes are the properties that define an entity. e.g.; **ROLL\_NO**, **NAME, ADDRESS**
* **Relation Schema:** A relation schema defines the structure of the relation and represents the name of the relation with its attributes. e.g.; STUDENT (ROLL\_NO, NAME, ADDRESS, PHONE, and AGE) is the relation schema for STUDENT. If a schema has more than 1 relation, it is called Relational Schema.
* **Tuple:** Each row in the relation is known as a tuple. The above relation contains 4 tuples, one of which is shown as:

| 1 | RAM | DELHI | 9455123451 | 18 |
| --- | --- | --- | --- | --- |

* **Relation Instance:** The set of tuples of a relation at a particular instance of time is called a relation instance. Table 1 shows the relation instance of STUDENT at a particular time. It can change whenever there is an insertion, deletion, or update in the database.
* **Degree:** The number of attributes in the relation is known as the degree of the relation. The **STUDENT** relation defined above has degree 5.
* **Cardinality:**The number of tuples in a relation is known as[cardinality](https://www.geeksforgeeks.org/cardinality-in-dbms/). The **STUDENT** relation defined above has cardinality 4.
* **Column:** The column represents the set of values for a particular attribute. The column **ROLL\_NO** is extracted from the relation STUDENT.

| **ROLL\_NO** |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |

* **NULL Values:** The value which is not known or unavailable is called a NULL value. It is represented by blank space. e.g.; PHONE of STUDENT having ROLL\_NO 4 is NULL.
* **Relation Key:**These are basically the keys that are used to identify the rows uniquely or also help in identifying tables. These are of the following types.
  + [Primary Key](https://www.geeksforgeeks.org/primary-key-constraint-in-sql/)
  + [Candidate Key](https://www.geeksforgeeks.org/difference-between-primary-and-candidate-key/)
  + [Super Key](https://www.geeksforgeeks.org/difference-between-super-key-and-candidate-key/)
  + [Foreign Key](https://www.geeksforgeeks.org/postgresql-foreign-key/)
  + [Alternate Key](https://www.geeksforgeeks.org/sql-alternate-key/)
  + [Composite Key](https://www.geeksforgeeks.org/composite-key-in-sql/)

## **Constraints in Relational Model**

While designing the Relational Model, we define some conditions which must hold for data present in the database are called Constraints. These constraints are checked before performing any operation (insertion, deletion, and updation ) in the database. If there is a violation of any of the constraints, the operation will fail.

### ****Domain Constraints****

These are attribute-level constraints. An attribute can only take values that lie inside the domain range. e.g.; If a constraint AGE>0 is applied to STUDENT relation, inserting a negative value of AGE will result in failure.

### ****Key Integrity****

Every relation in the database should have at least one set of attributes that defines a tuple uniquely. Those set of attributes is called keys. e.g.; ROLL\_NO in STUDENT is key. No two students can have the same roll number. So a key has two properties:

* It should be unique for all tuples.
* It can’t have NULL values.

### ****Referential Integrity****

When one attribute of a relation can only take values from another attribute of the same relation or any other relation, it is called [referential integrity](https://www.geeksforgeeks.org/cascading-referential-integrity-constraints-in-sql-server-management-studio/). Let us suppose we have 2 relations   
  
**Table Student**

| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** | **BRANCH\_CODE** |
| --- | --- | --- | --- | --- | --- |
| 1 | RAM | DELHI | 9455123451 | 18 | CS |
| 2 | RAMESH | GURGAON | 9652431543 | 18 | CS |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 | ECE |
| 4 | SURESH | DELHI |  | 18 | IT |

**Table Branch**

| **BRANCH\_CODE** | **BRANCH\_NAME** |
| --- | --- |
| **CS** | COMPUTER SCIENCE |
| **IT** | INFORMATION TECHNOLOGY |
| **ECE** | ELECTRONICS AND COMMUNICATION ENGINEERING |
| **CV** | CIVIL ENGINEERING |

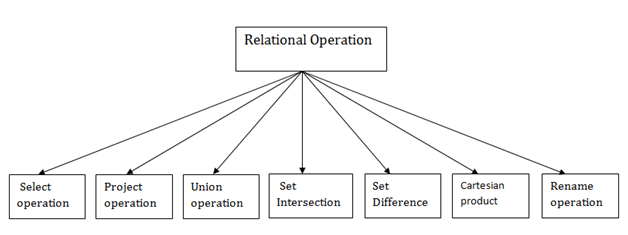
BRANCH\_CODE of STUDENT can only take the values which are present in BRANCH\_CODE of BRANCH which is called referential integrity constraint. The relation which is referencing another relation is called REFERENCING RELATION (STUDENT in this case) and the relation to which other relations refer is called REFERENCED RELATION (BRANCH in this case)

relational algebra

# Relational Algebra

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

## **Types of Relational operation**



### 1. Select Operation:

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

1. Notation:  σ p(r)

**Where:**

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

**For example: LOAN Relation**

|  |  |  |
| --- | --- | --- |
| **BRANCH\_NAME** | **LOAN\_NO** | **AMOUNT** |
| Downtown | L-17 | 1000 |
| Redwood | L-23 | 2000 |
| Perryride | L-15 | 1500 |
| Downtown | L-14 | 1500 |
| Mianus | L-13 | 500 |
| Roundhill | L-11 | 900 |
| Perryride | L-16 | 1300 |

**Input:**

1. σ BRANCH\_NAME="perryride" (LOAN)

**Output:**

|  |  |  |
| --- | --- | --- |
| **BRANCH\_NAME** | **LOAN\_NO** | **AMOUNT** |
| Perryride | L-15 | 1500 |
| Perryride | L-16 | 1300 |

### 2. Project Operation:

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

1. Notation: ∏ A1, A2, An (r)

**Where**

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

**Example: CUSTOMER RELATION**

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

**Input:**

ADVERTISEMENT

1. ∏ NAME, CITY (CUSTOMER)

**Output:**

|  |  |
| --- | --- |
| **NAME** | **CITY** |
| Jones | Harrison |
| Smith | Rye |
| Hays | Harrison |
| Curry | Rye |
| Johnson | Brooklyn |
| Brooks | Brooklyn |

### 3. Union Operation:

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

1. Notation: R ∪ S

A union operation must hold the following condition:

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

### Example:

**DEPOSITOR RELATION**

|  |  |
| --- | --- |
| **CUSTOMER\_NAME** | **ACCOUNT\_NO** |
| Johnson | A-101 |
| Smith | A-121 |
| Mayes | A-321 |
| Turner | A-176 |
| Johnson | A-273 |
| Jones | A-472 |
| Lindsay | A-284 |

**BORROW RELATION**

|  |  |
| --- | --- |
| **CUSTOMER\_NAME** | **LOAN\_NO** |
| Jones | L-17 |
| Smith | L-23 |
| Hayes | L-15 |
| Jackson | L-14 |
| Curry | L-93 |
| Smith | L-11 |
| Williams | L-17 |

**Input:**

1. ∏ CUSTOMER\_NAME (BORROW) ∪ ∏ CUSTOMER\_NAME (DEPOSITOR)

**Output:**

|  |
| --- |
| **CUSTOMER\_NAME** |
| Johnson |
| Smith |
| Hayes |
| Turner |
| Jones |
| Lindsay |
| Jackson |
| Curry |
| Williams |
| Mayes |

### 4. Set Intersection:

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

1. Notation: R ∩ S

**Example:** Using the above DEPOSITOR table and BORROW table

**Input:**

1. ∏ CUSTOMER\_NAME (BORROW) ∩ ∏ CUSTOMER\_NAME (DEPOSITOR)

**Output:**

|  |
| --- |
| **CUSTOMER\_NAME** |
| Smith |
| Jones |

### 5. Set Difference:

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

1. Notation: R - S

**Example:** Using the above DEPOSITOR table and BORROW table

**Input:**

1. ∏ CUSTOMER\_NAME (BORROW) - ∏ CUSTOMER\_NAME (DEPOSITOR)

**Output:**

|  |
| --- |
| **CUSTOMER\_NAME** |
| Jackson |
| Hayes |
| Willians |
| Curry |

### 6. Cartesian product

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

1. Notation: E X D

### Example:

**EMPLOYEE**

|  |  |  |
| --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_DEPT** |
| 1 | Smith | A |
| 2 | Harry | C |
| 3 | John | B |

**DEPARTMENT**

|  |  |
| --- | --- |
| **DEPT\_NO** | **DEPT\_NAME** |
| A | Marketing |
| B | Sales |
| C | Legal |

**Input:**

1. EMPLOYEE X DEPARTMENT

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_DEPT** | **DEPT\_NO** | **DEPT\_NAME** |
| 1 | Smith | A | A | Marketing |
| 1 | Smith | A | B | Sales |
| 1 | Smith | A | C | Legal |
| 2 | Harry | C | A | Marketing |
| 2 | Harry | C | B | Sales |
| 2 | Harry | C | C | Legal |
| 3 | John | B | A | Marketing |
| 3 | John | B | B | Sales |
| 3 | John | B | C | Legal |

### 7. Rename Operation:

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

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

1. ρ(STUDENT1, STUDENT)

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

tuple relational calculus

**Tuple Relational Calculus (TRC)**is a non-procedural query language used in relational database management systems (RDBMS) to retrieve data from tables. TRC is based on the concept of tuples, which 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](https://www.geeksforgeeks.org/what-is-database/), rather 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) }

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.

For example, let’s say we have a table called “Employees” with the following [attributes](https://www.geeksforgeeks.org/types-of-attributes-in-er-model/):

|  |
| --- |
| 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 TRC query:

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

In this query, the “Employees(t)” expression specifies that the tuple variable t represents a row in the “Employees” table. The “∧” symbol is the logical AND operator, which is used to combine the condition “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 employee 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 additional 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)](https://www.geeksforgeeks.org/structured-query-language/). However, it is useful in certain 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 Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains 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 all 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

## Domain Relational Calculus (DRC)

[Domain Relational Calculus](https://www.geeksforgeeks.org/domain-relational-calculus-in-dbms/) 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 |
| 1114 | ABC | 7000 |

**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 |
| 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.

{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 |

Unlock the Power of Placement Preparation!  
Feeling lost in OS, DBMS, CN, SQL, and DSA chaos? Our [Complete Interview Preparation](https://www.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses) Course is the ultimate guide to conquer placements. Trusted by over 100,000+ geeks, this course is your roadmap to interview triumph.

SQL: Characteristics of SQL

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

advantages of SQL

**Advantages of SQL :**   
SQL has many advantages which makes it popular and highly demanded. It is a reliable and efficient language used for communicating with the database. Some advantages of SQL are as follows: 

1. **Faster Query Processing –**   
   Large amount of data is retrieved quickly and efficiently. Operations like Insertion, deletion, manipulation of data is also done in almost no time.
2. **No Coding Skills –**   
   For data retrieval, large number of lines of code is not required. All basic keywords such as SELECT, INSERT INTO, UPDATE, etc are used and also the syntactical rules are not complex in SQL, which makes it a user-friendly language.
3. **Standardized Language –**   
   Due to documentation and long establishment over years, it provides a uniform platform worldwide to all its users.
4. **Portable –**   
   It can be used in programs in PCs, server, laptops independent of any platform (Operating System, etc). Also, it can be embedded with other applications as per need/requirement/use.
5. **Interactive Language –**   
   Easy to learn and understand, answers to complex queries can be received in seconds.

types of SQL commands

# SQL Commands

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

## **Types of SQL Commands**

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



SQL operators and their procedure

# SQL Operators

## **SQL Arithmetic Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Add | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_add) |
| - | Subtract | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_subtract) |
| \* | Multiply | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_multiply) |
| / | Divide | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_divide) |
| % | Modulo | [Try it](https://www.w3schools.com/sql/trymysql.asp?filename=trysql_op_modulo) |

## **SQL Bitwise Operators**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |

## **SQL Comparison Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Equal to | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_equal_to) |
| > | Greater than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than) |
| < | Less than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than) |
| >= | Greater than or equal to | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than2) |
| <= | Less than or equal to | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than2) |
| <> | Not equal to | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_not_equal_to) |

ADVERTISEMENT

## **SQL Compound Operators**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| += | Add equals |
| -= | Subtract equals |
| \*= | Multiply equals |
| /= | Divide equals |
| %= | Modulo equals |
| &= | Bitwise AND equals |
| ^-= | Bitwise exclusive equals |
| |\*= | Bitwise OR equals |

## **SQL Logical Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| ALL | TRUE if all of the subquery values meet the condition | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_all) |
| AND | TRUE if all the conditions separated by AND is TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_and) |
| ANY | TRUE if any of the subquery values meet the condition | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_any) |
| BETWEEN | TRUE if the operand is within the range of comparisons | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_between) |
| EXISTS | TRUE if the subquery returns one or more records | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_exists) |
| IN | TRUE if the operand is equal to one of a list of expressions | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in) |
| LIKE | TRUE if the operand matches a pattern | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_like) |
| NOT | Displays a record if the condition(s) is NOT TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_not) |
| OR | TRUE if any of the conditions separated by OR is TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_or) |
| SOME | TRUE if any of the subquery values meet the condition | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_some) |

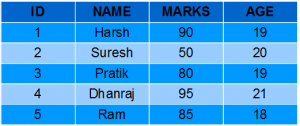
views

Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition. In this article we will learn about creating , deleting and updating Views.   
**Sample Tables**:

StudentDetails



StudentMarks



**CREATING VIEWS**

We can create View using **CREATE VIEW** statement. A View can be created from a single table or multiple tables. **Syntax**:

CREATE VIEW view\_name AS  
SELECT column1, column2.....  
FROM table\_name  
WHERE condition;

Index

# SQL Indexes

An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer. It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly.

An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements. Indexes can be created or dropped with no effect on the data. In this article, we will see how to [create](https://www.geeksforgeeks.org/sql-create-table/), [delete](https://www.geeksforgeeks.org/sql-delete-statement/), and use the INDEX in the database.

## **Creating an Index**

### ****Syntax****

***CREATE INDEX****index*

***ON TABLE****column;*

where the **index** is the name given to that index **TABLE** is the name of the table on which that index is created and **column** is the name of that column for which it is applied.

### ****For Multiple Columns****

**Syntax:**

***CREATE INDEX****index*

***ON TABLE****(column1, column2,…..);*

### ****For Unique Indexes****

Unique indexes are used for the maintenance of the integrity of the data present in the table as well as for fast performance, it does not allow multiple values to enter into the table.

**Syntax:**

***CREATE UNIQUE INDEX****index*

***ON TABLE****column;*

sub-queries

## **SQL Subqueries**

An SQL **Subquery**, is a SELECT query within another query. It is also known as **Inner query** or **Nested query** and the query containing it is the outer query.

The outer query can contain the SELECT, INSERT, UPDATE, and DELETE statements. We can use the subquery as a column expression, as a condition in SQL clauses, and with operators like =, >, <, >=, <=, IN, BETWEEN, etc.

## **Rules to be followed**

Following are the rules to be followed while writing subqueries −

* Subqueries must be enclosed within parentheses.
* Subqueries can be nested within another subquery.
* A subquery must contain the SELECT query and the FROM clause always.
* A subquery consists of all the clauses an ordinary SELECT clause can contain: GROUP BY, WHERE, HAVING, DISTINCT, TOP/LIMIT, etc. However, an ORDER BY clause is only used when a TOP clause is specified. It can't include COMPUTE or FOR BROWSE clause.
* A subquery can return a single value, a single row, a single column, or a whole table. They are called scalar subqueries.

## **Subqueries with the SELECT Statement**

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows −

SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

WHERE column\_name

OPERATOR (SELECT column\_name [,column\_name ] FROM table1 [, table2 ] [WHERE]);

aggregate functions

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.   **Various Aggregate Functions**

1) Count()

2) Sum()

3) Avg()

4) Min()

5) Max()

  Now let us understand each Aggregate function with a example:

Id Name Salary

-----------------------

1 A 80

2 B 40

3 C 60

4 D 70

5 E 60

6 F Null

**Count():**   ***Count(\*):*** Returns total number of records .i.e 6. ***Count(salary):*** Return number of Non Null values over the column salary. i.e 5. ***Count(Distinct Salary):*** Return number of distinct Non Null values over the column salary .i.e 4   **Sum():**   ***sum(salary):*** Sum all Non Null values of Column salary i.e., 310 ***sum(Distinct salary):*** Sum of all distinct Non-Null values i.e., 250.   **Avg():**   ***Avg(salary)*** = Sum(salary) / count(salary) = 310/5 ***Avg(Distinct salary)*** = sum(Distinct salary) / Count(Distinct Salary) = 250/4   **Min():**   ***Min(salary):*** Minimum value in the salary column except NULL i.e., 40. ***Max(salary):*** Maximum value in the salary i.e., 80.

Unlock the Power of Placement Preparation!  
Feeling lost in OS, DBMS, CN, SQL, and DSA chaos? Our [Complete Interview Preparation](https://www.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses) Course is the ultimate guide to conquer placements. Trusted by over 100,000+ geeks, this course is your roadmap to interview triumph.

Insert

## **The SQL INSERT INTO Statement**

The SQL **INSERT INTO Statement** is used to add new rows of data into a table in the database. Almost all the RDBMS provide this SQL query to add the records in database tables.

Each value in the records we are inserting in a table using this statement should be of the same datatype as the respective column and satisfy the constraints of the column (if any). The values passed using an insert statement should match the number of columns in the table or, the number of columns mentioned in the current query. If any of these conditions are not satisfied, this statement generates an error.

### Syntax

There are two basic syntaxes of the SQL **INSERT INTO** statement which are shown below −

INSERT INTO TABLE\_NAME (column1, column2...columnN)

VALUES (value1, value2...valueN);

update

The UPDATE statement in [SQL](https://www.geeksforgeeks.org/sql-tutorial/)is used to update the data of an existing table in the database. We can update single columns as well as multiple columns using the UPDATE statement as per our requirement.

In a very simple way, we can say that SQL commands(UPDATE and DELETE) are used to change the data that is already in the database. The SQL DELETE command uses a WHERE clause.

**Syntax**

*UPDATE table\_name SET column1 = value1, column2 = value2,…*

*WHERE condition;*

*table\_name: name of the table*

*column1: name of first , second, third column….*

*value1: new value for first, second, third column….*

*condition: condition to select the rows for which the*

*values of columns needs to be updated.*

### ****Parameter Explanation****

1. **UPDATE:** Command is used to update the column value in the table.
2. **WHERE:**Specifies the condition which we want to implement on the table.

**Note:**In the above query the**SET**statement is used to set new values to the particular column and the [**WHERE**](https://www.geeksforgeeks.org/sql-where-clause/)clause is used to select the rows for which the columns are needed to be updated. If we have not used the WHERE clause then the columns in all the rows will be updated. So the WHERE clause is used to choose the particular rows.

Let’s see the SQL update statement with examples.

**Query:**

CREATE TABLE Customer(  
 CustomerID INT PRIMARY KEY,  
 CustomerName VARCHAR(50),  
 LastName VARCHAR(50),  
 Country VARCHAR(50),  
 Age int(2),  
 Phone int(10)  
);

delete operations

SQL DELETE is a basic SQL operation used to delete data in a database. SQL DELETE is an important part of database management DELETE can be used to selectively remove records from a database table based on certain conditions. This SQL DELETE operation is important for database size management, data accuracy,and integrity.

**Syntax:**

*DELETE FROM table\_name*

*WHERE some\_condition;*

**Parameter Explanation**

* **Some\_condition**: condition to choose a particular record.
* **table\_name:** name of the table

joins

**SQL Join** statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

* INNER JOIN
* LEFT JOIN
* RIGHT JOIN
* FULL JOIN
* NATURAL JOIN

Consider the two tables below as follows:

**Student**



**StudentCourse**



The simplest Join is INNER JOIN.

### ****A. INNER JOIN****

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

**Syntax**:

SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1   
INNER JOIN table2  
ON table1.matching\_column = table2.matching\_column;  
  
  
**table1**: First table.  
**table2**: Second table  
**matching\_column**: Column common to both the tables.

***Note****: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.*



**Example Queries(INNER JOIN)**

This query will show the names and age of students enrolled in different courses.

SELECT StudentCourse.COURSE\_ID, Student.NAME, Student.AGE FROM Student  
INNER JOIN StudentCourse  
ON Student.ROLL\_NO = StudentCourse.ROLL\_NO;

**Output**:



### ****B. LEFT JOIN****

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.

**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1   
LEFT JOIN table2  
ON table1.matching\_column = table2.matching\_column;  
  
  
table1: First table.  
table2: Second table  
matching\_column: Column common to both the tables.

***Note****: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are the same.*



**Example Queries(LEFT JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID   
FROM Student  
LEFT JOIN StudentCourse   
ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output**:



### ****C. RIGHT JOIN****

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1   
RIGHT JOIN table2  
ON table1.matching\_column = table2.matching\_column;  
  
  
table1: First table.  
table2: Second table  
matching\_column: Column common to both the tables.

***Note****: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are the same.*



**Example Queries(RIGHT JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID   
FROM Student  
RIGHT JOIN StudentCourse   
ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**



### ****D. FULL JOIN****

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.



**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1   
FULL JOIN table2  
ON table1.matching\_column = table2.matching\_column;  
  
  
table1: First table.  
table2: Second table  
matching\_column: Column common to both the tables.

**Example Queries(FULL JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID   
FROM Student  
FULL JOIN StudentCourse   
ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**

| **NAME** | **COURSE\_ID** |
| --- | --- |
| HARSH | 1 |
| PRATIK | 2 |
| RIYANKA | 2 |
| DEEP | 3 |
| SAPTARHI | 1 |
| DHANRAJ | NULL |
| ROHIT | NULL |
| NIRAJ | NULL |
| NULL | 4 |
| NULL | 5 |
| NULL | 4 |

[Left JOIN (Video)](https://youtu.be/LCbO2U3jzU0)   
[Right JOIN (Video)](https://youtu.be/JOAe-yua6Jw)   
[Full JOIN (Video)](https://youtu.be/WmqAKSBupsE)   
[SQL | JOIN (Cartesian Join, Self Join)](https://www.geeksforgeeks.org/sql-join-cartesian-join-self-join/)

### E. Natural join (?)

Natural join can join tables based on the common columns in the tables being joined. A natural join returns all rows by matching values in common columns having same name and data type of columns and that column should be present in both tables.

Both table must have at list one common column with same column name and same data type.

The two table are joined using Cross join.

DBMS will look for a common column with same name and data type Tuples having exactly same values in common columns are kept in result.

Example:

| **Employee** | | |
| --- | --- | --- |
| **Emp\_id** | **Emp\_name** | **Dept\_id** |
| **1** | **Ram** | **10** |
| **2** | **Jon** | **30** |
| **3** | **Bob** | **50** |

| **Department** | |
| --- | --- |
| **Dept\_id** | **Dept\_name** |
| **10** | **IT** |
| **30** | **HR** |
| **40** | **TIS** |

Query: Find all Employees and their respective departments.

Solution: (Employee) ? (Department)

| **Emp\_id** | **Emp\_name** | **Dept\_id** | **Dept\_id** | **Dept\_name** |
| --- | --- | --- | --- | --- |
| **1** | **Ram** | **10** | **10** | **IT** |
| **2** | **Jon** | **30** | **30** | **HR** |
| **Employee data** | | | **Department data** | |

Unlock the Power of Placement Preparation!

union

The **UNION** operator could be used to find the result set or combination of two or more tables.

## **Terms and Conditions for using UNION**

* Each table used within UNION must have the same number of columns.
* The columns must have the same data types.
* The columns in each table must be in the same order.

### ****Syntax****

*SELECT columnnames FROM table1*

*UNION*

*SELECT columnnames FROM table2;*

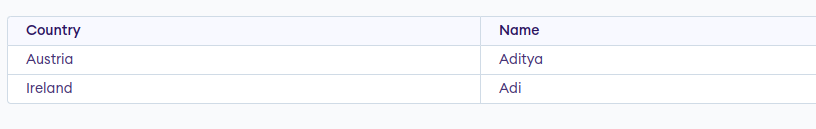
## **QL UNION ALL With WHERE**

The following [SQL](https://www.geeksforgeeks.org/what-is-sql/)statement returns the cities (duplicate values also) from both the “Geeks1” and the “Geeks2” tables:

### ****Query****

SELECT Country, Name FROM Emp1  
WHERE Name='Aditya'  
UNION ALL  
SELECT Country, Name FROM Emp2  
WHERE Country='Ireland'  
ORDER BY Country;

### ****Output****



intersection

The INTERSECT clause in SQL is used to combine two [SELECT](https://www.geeksforgeeks.org/sql-select-clause/) statements but the dataset returned by the INTERSECT statement will be the intersection of the data sets of the two SELECT statements. In simple words, the INTERSECT statement will return only those rows which will be common to both of the SELECT statements.



**Syntax**:

*SELECT column1 , column2 ….*

*FROM table\_names*

*WHERE condition*

***INTERSECT***

*SELECT column1 , column2 ….*

*FROM table\_names*

*WHERE condition*

Let’s assume that we have two table Customer Table and the Orders Table and we will perform some operations related to INTERSECT to understand better SQL intersect.

**Customers Table:**

CREATE TABLE Customer (

CustomerID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

Phone VARCHAR(20) NOT NULL,

Address VARCHAR(200) NOT NULL,

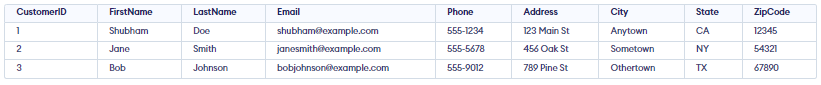
City VARCHAR(50) NOT NULL,

State VARCHAR(50) NOT NULL,

ZipCode VARCHAR(10) NOT NULL

);

**Output:**



**Orders Table:**

CREATE TABLE Orderss (

OrderID INT PRIMARY KEY,

CustomerID INT NOT NULL,

OrderDate DATE NOT NULL,

ShipDate DATE,

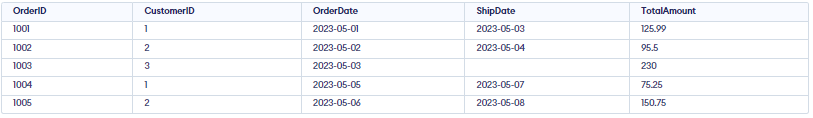
TotalAmount DECIMAL(10,2) NOT NULL,

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)

);

Let’s insert some random data in the order table

**Output:**



**Sample Queries**:

SELECT Customer.CustomerID, Customer.FirstName, Customer.LastName

FROM Customer

LEFT JOIN Orderss ON Customer.CustomerID = Orderss.CustomerID

UNION

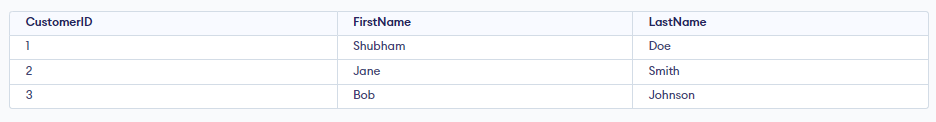
SELECT Customer.CustomerID, Customer.FirstName, Customer.LastName

FROM Customer

LEFT JOIN Orderss ON Customer.CustomerID = Orderss.CustomerID

WHERE Orderss.OrderID IS NULL;

**Output:**



## INTERSECT with BETWEEN Operator

As we have already discussed in the initial syntax, we can also use the INTERSECT operator along with some conditional operators. We can use the INTERSECT operator with the BETWEEN operator in SQL to find rows that fall within a specified range.

Let’s assume that we have one table name Customer and another one as orderss.

**Query:**

SELECT CustomerID, FirstName, LastName

FROM Customer

WHERE CustomerID BETWEEN 100 AND 200

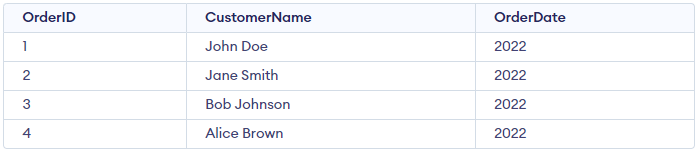
INTERSECT

SELECT CustomerID, FirstName, LastName

FROM Customer

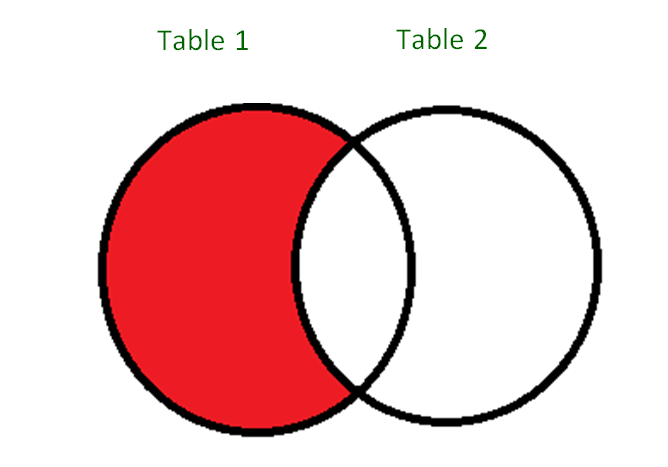
WHERE LastName BETWEEN 'A' AND 'M';

**Output:**



minus

**SQL | MINUS Operator**

The Minus Operator in SQL is used with two SELECT statements. The MINUS operator is used to subtract the result set obtained by first SELECT query from the result set obtained by second SELECT query. In simple words, we can say that MINUS operator will return only those rows which are unique in only first SELECT query and not those rows which are common to both first and second SELECT queries. **Pictorial Representation**:[](https://media.geeksforgeeks.org/wp-content/uploads/SQL_Minus_Operator_1.png)As you can see is in the above diagram, the MINUS operator will return only those rows which are present in the result set from Table1 and not present in the result set of Table2. **Basic Syntax**:

SELECT column1 , column2 , ... columnN

FROM table\_name

WHERE condition

MINUS

SELECT column1 , column2 , ... columnN

FROM table\_name

WHERE condition;

cursors in SQL

# Cursor in SQL Server

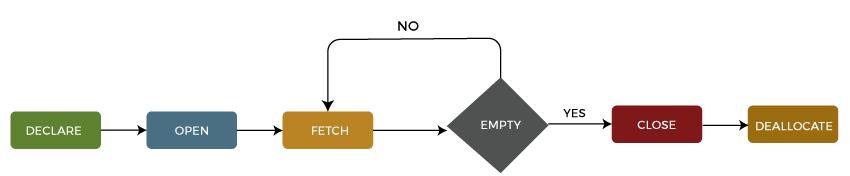
A cursor in SQL Server is a d**atabase object that allows us to retrieve each row at a time and manipulate its data**. A cursor is nothing more than a pointer to a row. It's always used in conjunction with a SELECT statement. It is usually a collection of [SQL](https://www.javatpoint.com/sql-tutorial) logic that loops through a predetermined number of rows one by one. A simple illustration of the cursor is when we have an extensive database of worker's records and want to calculate each worker's salary after deducting taxes and leaves.

The [SQL Server](https://www.javatpoint.com/sql-server-tutorial) **cursor's purpose is to update the data row by row, change it, or perform calculations that are not possible when we retrieve all records at once**. It's also useful for performing administrative tasks like SQL Server database backups in sequential order. Cursors are mainly used in the development, DBA, and ETL processes.

This article explains everything about SQL Server cursor, such as cursor life cycle, why and when the cursor is used, how to implement cursors, its limitations, and how we can replace a cursor.

## **Life Cycle of the cursor**

We can describe the life cycle of a cursor into the **five different sections** as follows:



### 1: Declare Cursor

The first step is to declare the cursor using the below SQL statement:

1. **DECLARE** cursor\_name **CURSOR**
2. **FOR** select\_statement;

We can declare a cursor by specifying its name with the data type CURSOR after the DECLARE keyword. Then, we will write the SELECT statement that defines the output for the cursor.

### 2: Open Cursor

It's a second step in which we open the cursor to store data retrieved from the result set. We can do this by using the below SQL statement:

1. **OPEN** cursor\_name;

### 3: Fetch Cursor

It's a third step in which rows can be fetched one by one or in a block to do data manipulation like insert, update, and delete operations on the currently active row in the cursor. We can do this by using the below SQL statement:

1. **FETCH** **NEXT** **FROM** **cursor** **INTO** variable\_list;

We can also use the **@@FETCHSTATUS function** in SQL Server to get the status of the most recent FETCH statement cursor that was executed against the cursor. The **FETCH** statement was successful when the @@FETCHSTATUS gives zero output. The **WHILE** statement can be used to retrieve all records from the cursor. The following code explains it more clearly:

1. WHILE @@FETCH\_STATUS = 0
2. **BEGIN**
3. **FETCH** **NEXT** **FROM** cursor\_name;
4. **END**;

### 4: Close Cursor

It's a fourth step in which the cursor should be closed after we finished work with a cursor. We can do this by using the below SQL statement:

1. **CLOSE** cursor\_name;

### 5: Deallocate Cursor

It is the fifth and final step in which we will erase the cursor definition and release all the system resources associated with the cursor. We can do this by using the below SQL statement:

1. **DEALLOCATE** cursor\_name;

## **Uses of SQL Server Cursor**

We know that relational database management systems, including SQL Server, are excellent in handling data on a set of rows called result sets. **For example**, we have a table **product\_table** that contains the product descriptions. If we want to update the **price** of the product, then the below '**UPDATE'** query will update all records that match the condition in the '**WHERE'** clause:

1. **UPDATE** product\_table **SET** unit\_price = 100 **WHERE** product\_id = 105;

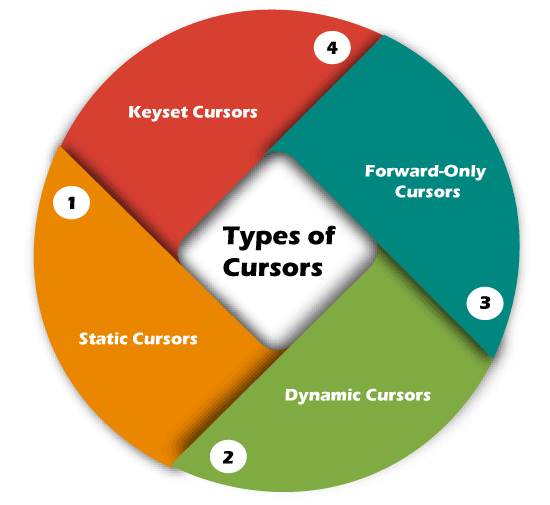
Sometimes the application needs to process the rows in a singleton fashion, i.e., on row by row basis rather than the entire result set at once. We can do this process by using cursors in SQL Server. Before using the cursor, we must know that cursors are very bad in performance, so it should always use only when there is no option except the cursor.

The cursor uses the same technique as we use loops like FOREACH, FOR, WHILE, DO WHILE to iterate one object at a time in all programming languages. Hence, it could be chosen because it applies the same logic as the programming language's looping process.

## **Types of Cursors in SQL Server**

The following are the different types of cursors in SQL Server listed below:

* Static Cursors
* Dynamic Cursors
* Forward-Only Cursors
* Keyset Cursors



### Static Cursors

The result set shown by the static cursor is always the same as when the cursor was first opened. Since the static cursor will store the result in **tempdb**, they are always **read-only**. We can use the static cursor to move both forward and backward. In contrast to other cursors, it is slower and consumes more memory. As a result, we can use it only when scrolling is necessary, and other cursors aren't suitable.

This cursor shows rows that were removed from the database after it was opened. A static cursor does not represent any INSERT, UPDATE, or DELETE operations (unless the cursor is closed and reopened).

### Dynamic Cursors

The dynamic cursors are opposite to the static cursors that allow us to perform the data updation, deletion, and insertion operations while the cursor is open. It is **scrollable by default**. It can detect all changes made to the rows, order, and values in the result set, whether the changes occur inside the cursor or outside the cursor. Outside the cursor, we cannot see the updates until they are committed.

### Forward-Only Cursors

It is the default and fastest cursor type among all cursors. It is called a forward-only cursor because it **moves only forward through the result set**. This cursor doesn't support scrolling. It can only retrieve rows from the beginning to the end of the result set. It allows us to perform insert, update, and delete operations. Here, the effect of insert, update and delete operations made by the user that affect rows in the result set are visible as the rows are fetched from the cursor. When the row was fetched, we cannot see the changes made to rows through the cursor.

Domain constraints

**Domain constraints in DBMS**

In DBMS, constraints are the set of rules that ensures that when an authorized user modifies the database they do not disturb the data consistency and the constraints are specified within the DDL commands like “alter” and “create” command. There are several types of constraints available in DBMS and they are:

* Domain constraints
* Entity Integrity constraints
* Referential Integrity constraints
* Key constraints

In this article, we will only discuss domain constraints.

**Domain Constraints**

Domain Constraints are user-defined columns that help the user to enter the value according to the data type. And if it encounters a wrong input it gives the message to the user that the column is not fulfilled properly. Or in other words, it is an attribute that specifies all the possible values that the attribute can hold like integer, character, date, time, string, etc. It defines the domain or the set of values for an attribute and ensures that the value taken by the attribute must be an atomic value(Can’t be divided) from its domain.

Domain Constraint = data type(integer / character/date / time / string / etc.) +

Constraints(NOT NULL / UNIQUE / PRIMARY KEY /

FOREIGN KEY / CHECK / DEFAULT)

**Type of domain constraints:**

There are two types of constraints that come under domain constraint and they are:

**1. Domain Constraints – Not Null:**Null values are the values that are unassigned or we can also say that which are unknown or the missing attribute values and by default, a column can hold the null values. Now as we know that the Not Null constraint restricts a column to not accept the null values which means it only restricts a field to always contain a value which means you cannot insert a new record or update a record without adding a value into the field.

**Example:**In the ’employee’ database, every employee must have a name associated with them.

Create table employee

(employee\_id varchar(30),

employee\_name varchar(30) not null,

salary NUMBER);

**2. Domain Constraints – Check:**It defines a condition that each row must satisfy which means it restricts the value of a column between ranges or we can say that it is just like a condition or filter checking before saving data into a column. It ensures that when a tuple is inserted inside the relation must satisfy the predicate given in the check clause.

**Example:**We need to check whether the entered id number is greater than 0 or not for the employee table.

Create table employee

(employee\_id varchar(30) not null check(employee\_id > 0),

employee\_name varchar(30),

salary NUMBER);

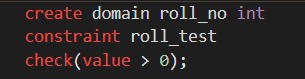
The above example creates CHECK constraints on the employee\_id column and specifies that the column employee\_id must only include integers greater than 0.

**Note:** In DBMS a table is a combination of rows and columns in which we have some unique attribute names associated with it. And basically, a domain is a unique set of values present in a table. Let’s take an example, suppose we have a table student which consists of 3 attributes as NAME, ROLL NO, and MARKS. Now ROLL NO attributes can have only numbers associated with them and they won’t contain any alphabet. So we can say that it contains the domain of integer only and it can be only a positive number greater than 0.

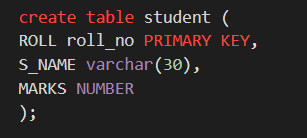
**Example 1:**

Creating a table “student” with the “ROLL” field having a value greater than 0.

**Domain:**



**Table:**

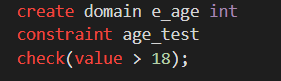


The above example will only accept the roll no. which is greater than 0.

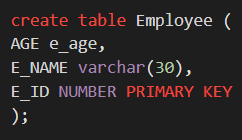
**Example 2:**

Creating a table “Employee” with the “AGE” field having a value greater than 18.

**Domain:**



**Table:**



referential integrity

Referential Integrity constraints in DBMS

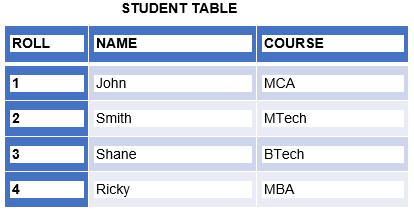
A referential integrity constraint is also known as **foreign key constraint**. A foreign key is a key whose values are derived from the Primary key of another table.

The table from which the values are derived is known as **Master or Referenced** Table and the Table in which values are inserted accordingly is known as **Child or Referencing** Table, In other words, we can say that the table containing the **foreign key** is called the **child table**, and the table containing the **Primary key/candidate key** is called the **referenced or parent table**. When we talk about the database relational model, the candidate key can be defined as a set of attribute which can have zero or more attributes.

**The syntax of the Master Table or Referenced table is:**

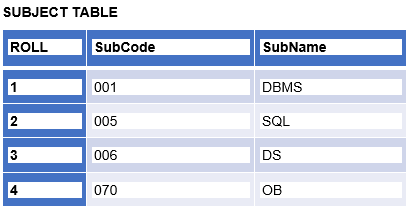
1. CREATE TABLE Student (Roll **int** PRIMARY KEY, Name varchar(25) , Course varchar(10) );

Here column Roll is acting as **Primary Key,** which will help in deriving the value of foreign key in the child table.



**The syntax of Child Table or Referencing table is:**

1. CREATE TABLE Subject (Roll **int** references Student, SubCode **int**, SubName varchar(10) );



In the above table, column Roll is acting as **Foreign Key,** whose values are derived using the Roll value of Primary key from Master table.

Assertions

An assertion is a predicate expressing a condition that we wish the database always to satisfy. ♣ The following constraints, can be expressed using assertions: ♣ For each tuple in the student relation, the value of the attribute tot\_cred must equal the sum of credits of courses that the student has completed successfully. ♣ An instructor cannot teach in two different classrooms in a semester in the same time slot ♣ An assertion in SQL takes the form: create assertion check ();

Triggers

# QL Trigger | Student Database

A trigger is a stored procedure in a database that automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when specific table columns are updated in simple words a trigger is a collection of [SQL](https://www.geeksforgeeks.org/sql-tutorial/)statements with particular names that are stored in system memory. It belongs to a specific class of stored procedures that are automatically invoked in response to database server events. Every trigger has a table attached to it.

Because a trigger cannot be called directly, unlike a stored procedure, it is referred to as a special procedure. A trigger is automatically called whenever a data modification event against a table takes place, which is the main distinction between a trigger and a procedure. On the other hand, a stored procedure must be called directly.

The following are the key differences between triggers and stored procedures:

1. Triggers cannot be manually invoked or executed.
2. There is no chance that triggers will receive parameters.
3. A transaction cannot be committed or rolled back inside a trigger.

**Syntax:**

*create trigger [trigger\_name]*

*[before | after]*

*{insert | update | delete}*

*on [table\_name]*

*[for each row]*

*[trigger\_body]*

### ****Explanation of Syntax****

## **Advantage of Triggers**

The benefits of using triggers in SQL Server include the following:

1. Database object rules are established by triggers, which cause changes to be undone if they are not met.
2. The trigger will examine the data and, if necessary, make changes.
3. We can enforce data integrity thanks to triggers.
4. Data is validated using triggers before being inserted or updated.
5. Triggers assist us in maintaining a records log.
6. Due to the fact that they do not need to be compiled each time they are run, triggers improve the performance of SQL queries.
7. The client-side code is reduced by triggers, saving time and labor.
8. Trigger maintenance is simple.

## **Disadvantage**of Triggers

The drawbacks of using triggers in SQL Server include the following:

1. Only triggers permit the use of extended validations.
2. Automatic triggers are used, and the user is unaware of when they are being executed. Consequently, it is difficult to troubleshoot issues that arise in the database layer.
3. The database server’s overhead may increase as a result of triggers.
4. In a single CREATE TRIGGER statement, we can specify the same trigger action for multiple user actions, such as INSERT and UPDATE.
5. Only the current database is available for creating triggers, but they can still make references to objects outside the database.

authorization and authentication

Let us see the difference between authentication and authorization:

| **Authentication** | **Authorization** |
| --- | --- |
| In the [authentication](https://www.geeksforgeeks.org/authentication-in-computer-network/) process, the identity of users are checked for providing the access to the system. | While in [authorization](https://www.geeksforgeeks.org/what-is-aaa-authentication-authorization-and-accounting/) process, a the person’s or user’s authorities are checked for accessing the resources. |
| In the authentication process, users or persons are verified. | While in this process, users or persons are validated. |
| It is done before the authorization process. | While this process is done after the authentication process. |
| It needs usually the user’s login details. | While it needs the user’s privilege or security levels. |
| Authentication determines whether the person is user or not. | While it determines **What permission does the user have?** |
| Generally, transmit information through an ID Token. | Generally, transmit information through an Access Token. |
| The OpenID Connect (OIDC) protocol is an authentication protocol that is generally in charge of user authentication process. | The OAuth 2.0 protocol governs the overall system of user authorization process. |
| Popular Authentication Techniques-   * Password-Based Authentication * Passwordless Authentication * 2FA/MFA (Two-Factor Authentication / Multi-Factor Authentication) * [Single sign-on (SSO)](https://www.geeksforgeeks.org/introduction-of-single-sign-on-sso/) * Social authentication | Popular  Authorization Techniques-   * Role-Based Access Controls (RBAC) * [JSON web token (JWT) Authorization](https://www.geeksforgeeks.org/json-web-token-jwt/) * SAML Authorization * OpenID Authorization * OAuth 2.0 Authorization |
| The authentication credentials can be changed in part as and when required by the user. | The authorization permissions cannot be changed by user as these are granted by the owner of the system and only he/she has the access to change it. |
| The user authentication is visible at user end. | The user authorization is not visible at the user end. |
| The user authentication is identified with username, password, face recognition, retina scan, fingerprints, etc. | The user authorization is carried out through the access rights to resources by using roles that have been pre-defined. |
| **Example**: Employees in a company are required to authenticate through the network before accessing their company email. | **Example:** After an employee successfully authenticates, the system determines what information the employees are allowed to access. |

normalization

* Normalization is the process of organizing the data in the database.
* Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
* Normalization divides the larger table into smaller and links them using relationships.
* The normal form is used to reduce redundancy from the database table.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

Functional dependencies

# Functional Dependency

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

1. X   →   Y

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

**For example:**

Assume we have an employee table with attributes: Emp\_Id, Emp\_Name, Emp\_Address.

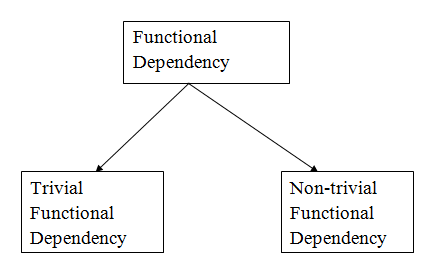
Here Emp\_Id attribute can uniquely identify the Emp\_Name attribute of employee table because if we know the Emp\_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp\_Id → Emp\_Name

We can say that Emp\_Name is functionally dependent on Emp\_Id.

## **Types of Functional dependency**



### 1. Trivial functional dependency

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

**Example:**

1. Consider a table with two columns Employee\_Id and Employee\_Name.
2. {Employee\_id, Employee\_Name}   →    Employee\_Id is a trivial functional dependency as
3. Employee\_Id is a subset of {Employee\_Id, Employee\_Name}.
4. Also, Employee\_Id → Employee\_Id and Employee\_Name   →    Employee\_Name are trivial dependencies too.

### 2. Non-trivial functional dependency

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

**Example:**

1. ID   →    Name,
2. Name   →    DOB

normal forms- First

## **Types of Normal Forms:**

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

**Following are the various types of Normal forms:**



|  |  |
| --- | --- |
| **Normal Form** | **Description** |
| [1NF](https://www.javatpoint.com/dbms-first-normal-form) | A relation is in 1NF if it contains an atomic value. |
| [2NF](https://www.javatpoint.com/dbms-second-normal-form) | A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key. |
| [3NF](https://www.javatpoint.com/dbms-third-normal-form) | A relation will be in 3NF if it is in 2NF and no transition dependency exists. |
| BCNF | A stronger definition of 3NF is known as Boyce Codd's normal form. |
| [4NF](https://www.javatpoint.com/dbms-forth-normal-form) | A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency. |
| [5NF](https://www.javatpoint.com/dbms-fifth-normal-form) | A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless. |

## **Advantages of Normalization**

* Normalization helps to minimize data redundancy.
* Greater overall database organization.
* Data consistency within the database.
* Much more flexible database design.
* Enforces the concept of relational integrity.

## **Disadvantages of Normalization**

* You cannot start building the database before knowing what the user needs.
* The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
* It is very time-consuming and difficult to normalize relations of a higher degree.
* Careless decomposition may lead to a bad database design, leading to serious problem

First normal form

First Normal Form (1NF)

* A relation will be 1NF if it contains an atomic value.
* It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
* First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

**Example:** Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP\_PHONE.

**EMPLOYEE table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385, 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389, 8589830302 | Punjab |

The decomposition of the EMPLOYEE table into 1NF has been shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385 | UP |
| 14 | John | 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389 | Punjab |
| 12 | Sam | 8589830302 | Punjab |

second

# Second Normal Form (2NF)

* In the 2NF, relational must be in 1NF.
* In the second normal form, all non-key attributes are fully functional dependent on the primary key

**Example:** Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

**TEACHER table**

|  |  |  |
| --- | --- | --- |
| **TEACHER\_ID** | **SUBJECT** | **TEACHER\_AGE** |
| 25 | Chemistry | 30 |
| 25 | Biology | 30 |
| 47 | English | 35 |
| 83 | Math | 38 |
| 83 | Computer | 38 |

In the given table, non-prime attribute TEACHER\_AGE is dependent on TEACHER\_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

**TEACHER\_DETAIL table:**

|  |  |
| --- | --- |
| **TEACHER\_ID** | **TEACHER\_AGE** |
| 25 | 30 |
| 47 | 35 |
| 83 | 38 |

**TEACHER\_SUBJECT table:**

|  |  |
| --- | --- |
| **TEACHER\_ID** | **SUBJECT** |
| 25 | Chemistry |
| 25 | Biology |
| 47 | English |
| 83 | Math |
| 83 | Computer |

third

Third Normal Form (3NF)

* A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
* 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
* If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

**Example:**

**EMPLOYEE\_DETAIL table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** |
| 222 | Harry | 201010 | UP | Noida |
| 333 | Stephan | 02228 | US | Boston |
| 444 | Lan | 60007 | US | Chicago |
| 555 | Katharine | 06389 | UK | Norwich |
| 666 | John | 462007 | MP | Bhopal |

**Super key in the table above:**

* 1. {EMP\_ID}, {EMP\_ID, EMP\_NAME}, {EMP\_ID, EMP\_NAME, EMP\_ZIP}....so on

**Candidate key:** {EMP\_ID}

**Non-prime attributes:** In the given table, all attributes except EMP\_ID are non-prime.

Here, EMP\_STATE & EMP\_CITY dependent on EMP\_ZIP and EMP\_ZIP dependent on EMP\_ID. The non-prime attributes (EMP\_STATE, EMP\_CITY) transitively dependent on super key(EMP\_ID). It violates the rule of third normal form.

That's why we need to move the EMP\_CITY and EMP\_STATE to the new <EMPLOYEE\_ZIP> table, with EMP\_ZIP as a Primary key.

**EMPLOYEE table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** |
| 222 | Harry | 201010 |
| 333 | Stephan | 02228 |
| 444 | Lan | 60007 |
| 555 | Katharine | 06389 |
| 666 | John | 462007 |

**EMPLOYEE\_ZIP table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** |
| 201010 | UP | Noida |
| 02228 | US | Boston |
| 60007 | US | Chicago |
| 06389 | UK | Norwich |
| 462007 | MP | Bhopal |

BCNF

Boyce Codd normal form (BCNF)

* BCNF is the advance version of 3NF. It is stricter than 3NF.
* A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
* For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

**Example:** Let's assume there is a company where employees work in more than one department.

**EMPLOYEE table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_COUNTRY** | **EMP\_DEPT** | **DEPT\_TYPE** | **EMP\_DEPT\_NO** |
| 264 | India | Designing | D394 | 283 |
| 264 | India | Testing | D394 | 300 |
| 364 | UK | Stores | D283 | 232 |
| 364 | UK | Developing | D283 | 549 |

**In the above table Functional dependencies are as follows:**

1. EMP\_ID  →  EMP\_COUNTRY
2. EMP\_DEPT  →   {DEPT\_TYPE, EMP\_DEPT\_NO}

**Candidate key: {EMP-ID, EMP-DEPT}**

The table is not in BCNF because neither EMP\_DEPT nor EMP\_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

**EMP\_COUNTRY table:**

|  |  |
| --- | --- |
| **EMP\_ID** | **EMP\_COUNTRY** |
| 264 | India |
| 264 | India |

**EMP\_DEPT table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_DEPT** | **DEPT\_TYPE** | **EMP\_DEPT\_NO** |
| Designing | D394 | 283 |
| Testing | D394 | 300 |
| Stores | D283 | 232 |
| Developing | D283 | 549 |

**EMP\_DEPT\_MAPPING table:**

|  |  |
| --- | --- |
| **EMP\_ID** | **EMP\_DEPT** |
| D394 | 283 |
| D394 | 300 |
| D283 | 232 |
| D283 | 549 |

**Functional dependencies:**

EMP\_ID   →    EMP\_COUNTRY

1. EMP\_DEPT   →   {DEPT\_TYPE, EMP\_DEPT\_NO}

**Candidate keys:**

**For the first table:** EMP\_ID  
**For the second table:** EMP\_DEPT  
**For the third table:** {EMP\_ID, EMP\_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Fourth

Fourth normal form (4NF)

* A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
* For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

**STUDENT**

|  |  |  |
| --- | --- | --- |
| **STU\_ID** | **COURSE** | **HOBBY** |
| 21 | Computer | Dancing |
| 21 | Math | Singing |
| 34 | Chemistry | Dancing |
| 74 | Biology | Cricket |
| 59 | Physics | Hockey |

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU\_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU\_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

**STUDENT\_COURSE**

|  |  |
| --- | --- |
| **STU\_ID** | **COURSE** |
| 21 | Computer |
| 21 | Math |
| 34 | Chemistry |
| 74 | Biology |
| 59 | Physics |

**STUDENT\_HOBBY**

|  |  |
| --- | --- |
| **STU\_ID** | **HOBBY** |
| 21 | Dancing |
| 21 | Singing |
| 34 | Dancing |
| 74 | Cricket |
| 59 | Hockey |

fifth normal forms

Fifth normal form (5NF)

* A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
* 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
* 5NF is also known as Project-join normal form (PJ/NF).

Example

|  |  |  |
| --- | --- | --- |
| **SUBJECT** | **LECTURER** | **SEMESTER** |
| Computer | Anshika | Semester 1 |
| Computer | John | Semester 1 |
| Math | John | Semester 1 |
| Math | Akash | Semester 2 |
| Chemistry | Praveen | Semester 1 |

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

**P1**

|  |  |
| --- | --- |
| **SEMESTER** | **SUBJECT** |
| Semester 1 | Computer |
| Semester 1 | Math |
| Semester 1 | Chemistry |
| Semester 2 | Math |

**P2**

|  |  |
| --- | --- |
| **SUBJECT** | **LECTURER** |
| Computer | Anshika |
| Computer | John |
| Math | John |
| Math | Akash |
| Chemistry | Praveen |

**P3**

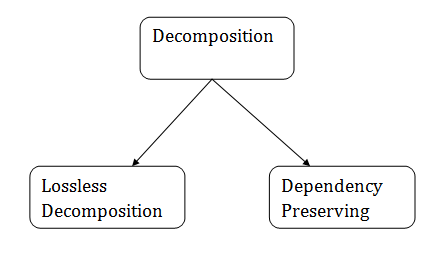
|  |  |
| --- | --- |
| **SEMSTER** | **LECTURER** |
| Semester 1 | Anshika |
| Semester 1 | John |
| Semester 1 | John |
| Semester 2 | Akash |
| Semester 1 | Praveen |

decomposition

# Relational Decomposition

* When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
* In a database, it breaks the table into multiple tables.
* If the relation has no proper decomposition, then it may lead to problems like loss of information.
* Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

## **Types of Decomposition**



### Lossless Decomposition

* If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
* The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
* The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

**Example:**

**EMPLOYEE\_DEPARTMENT table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** | **DEPT\_ID** | **DEPT\_NAME** |
| 22 | Denim | 28 | Mumbai | 827 | Sales |
| 33 | Alina | 25 | Delhi | 438 | Marketing |
| 46 | Stephan | 30 | Bangalore | 869 | Finance |
| 52 | Katherine | 36 | Mumbai | 575 | Production |
| 60 | Jack | 40 | Noida | 678 | Testing |

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

**EMPLOYEE table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** |
| 22 | Denim | 28 | Mumbai |
| 33 | Alina | 25 | Delhi |
| 46 | Stephan | 30 | Bangalore |
| 52 | Katherine | 36 | Mumbai |
| 60 | Jack | 40 | Noida |

**DEPARTMENT table**

|  |  |  |
| --- | --- | --- |
| **DEPT\_ID** | **EMP\_ID** | **DEPT\_NAME** |
| 827 | 22 | Sales |
| 438 | 33 | Marketing |
| 869 | 46 | Finance |
| 575 | 52 | Production |
| 678 | 60 | Testing |

Now, when these two relations are joined on the common column "EMP\_ID", then the resultant relation will look like:

**Employee ⋈ Department**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** | **DEPT\_ID** | **DEPT\_NAME** |
| 22 | Denim | 28 | Mumbai | 827 | Sales |
| 33 | Alina | 25 | Delhi | 438 | Marketing |
| 46 | Stephan | 30 | Bangalore | 869 | Finance |
| 52 | Katherine | 36 | Mumbai | 575 | Production |
| 60 | Jack | 40 | Noida | 678 | Testing |

Hence, the decomposition is Lossless join decomposition.

### Dependency Preserving

* It is an important constraint of the database.
* In the dependency preservation, at least one decomposed table must satisfy every dependency.
* If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
* For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).