**UNIT-2**

**Introduction to relational model**

The main construct for representing data in the relational model is a **relation**. A relation consists of a **relation schema** and a **relation instance**. The relation instanceis a table, and the relation schema describes the column heads for the table. We first describe the relation schema and then the relation instance. The schema specifies the relation’s name, the name of each **field** (or **column**, or **attribute**), and the **domain** of each field. A domain is referred to in a relation schema by the **domain name** and has a set of associated **values**.

**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 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**

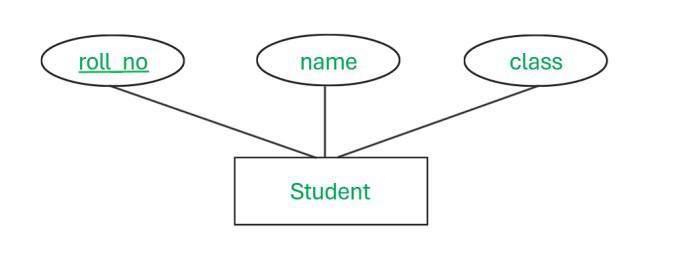
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| --- | --- | --- | --- | --- |
| 1 | RAM | DELHI | 9455123451 | 18 |
| 2 | RAMESH | GURGAON | 9652431543 | 18 |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |
| 4 | SURESH | DELHI | 9456253131 | 18 |

**Attributes**

Attributes are properties or characteristics of an entity. Attributes are used to describe the entity. The attribute is nothing but a piece of data that gives more information about the entity. Attributes are used to distinguish one entity from the other entity. Attributes help to categorize the entity and the entity can be easily retrieved and manipulate the entity. Attributes can help the database to be more structural and hierarchical. An entity with no attribute is of no use in the database.

## Example

* Let’s take the student as an entity. Students will have multiple attributes such as roll number, name, and class.
* These attributes are used to describe the student in more detail.



* As shown in the figure, roll\_no, name, and class are the attributes of the entity Student.
* All three attributes give meaning to the entity. The information about the student entity lies in all 3 attributes.

## Types Of Attribute

There are 8 types of attributes in [DBMS](https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1/" \t "_blank).

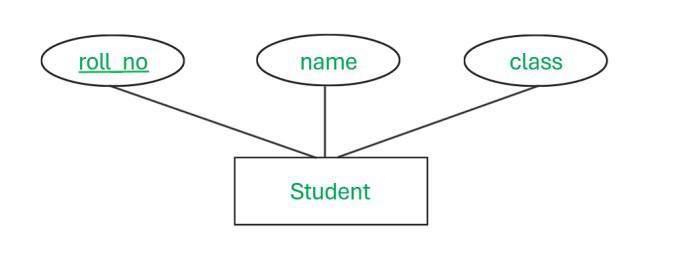
* **Simple Attribute.**
* **Composite Attribute.**
* **Single Valued Attribute.**
* **Multivalued Attribute.**
* **Key Attribute.**
* **Derived Attribute.**
* **Stored Attribute.**
* **Complex Attribute.**

## Simple Attribute

* Simple attributes are those attributes that cannot be divided into more attributes. Simple attributes state the simple information about the entity such as name, roll\_no, class, age, etc.
* Simple attributes are widely used for storing information about the entity.

### Example

roll\_no, class, and name

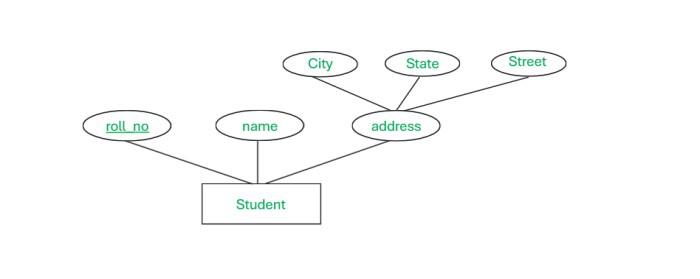


## Composite Attribute

* When 2 or more than 2 simple attributes are combined to make an attribute then that attribute is called a **Composite attribute.**
* The composite attribute is made up of multiple attributes. After combining these attributes, the composed attributes are formed.
* Complex attributes are used where data is complex and needs to be stored in a complex structure.

### Example

City, State, and Street.

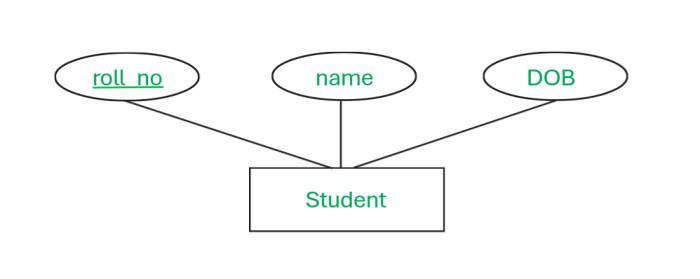


## Single Valued Attribute

* The attribute with only a single value is known as a **single-valued attribute.**These attributes have a single value for each instance of a given entity.
* Mostly these attributes are used to provide the unique identity to the multiple instances of attributes.

### Example

DOB

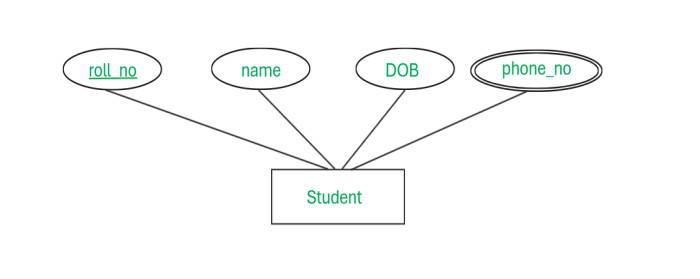


## Multivalued Attribute

* An attribute which can have multiple values is known as a **multivalued attribute.**Multivalued attributes have multiple values for the single instance of an entity.
* Keu of entity is associated with multiple values. It does not have only one value. It is the opposite of the single-valued attribute.

### Example

 phone\_no

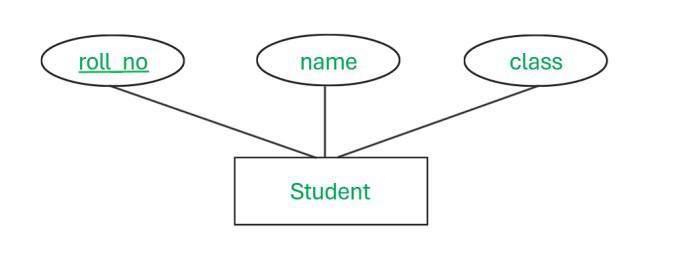


## Key Attribute

* The attribute which has unique values for every row in the table is known as a **Key Attribute**. The key attribute has a very crucial role in the database.
* The key attribute is a distinct and unique characteristic of the entity that can be used to identify the entity uniquely.

### Example

roll\_no.

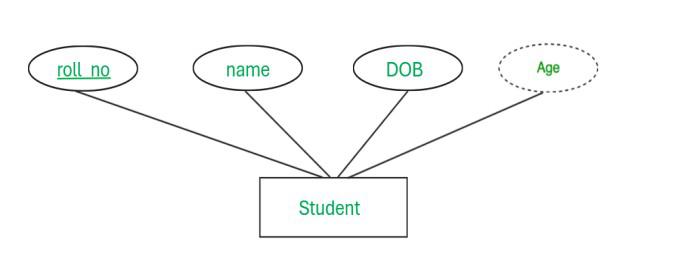


## Derived Attribute

* The attribute that can be derived from the other attributes and does not require to be already present in the database is called a **Derived Attribute**.
* Derived attributes are not stored in the Database directly. It is calculated by using the stored attributes in the database.

### Example

* Here the student has multiple attributes including DOB and age. It is observed that age can be calculated with the help of the DOB attribute.
* So age is a derived attribute that is derived from an attribute named DOB.

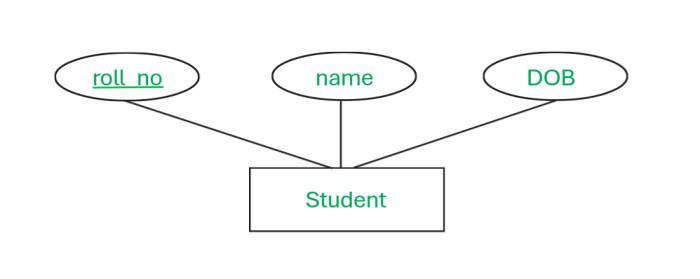


## Stored Attribute

* If the data of the attribute remains constant for every instance of entity then it is called a **[Stored Attribute.](https://www.geeksforgeeks.org/difference-between-stored-and-derived-attribute/" \t "_blank)**
* The value of the attribute present in the database does not get updated and it remains constant once it is stored.
* These attributes are used to store permanent information about an entity which will remain constant throughout the lifetime of the entity.

### Example

* The student has 3 attributes as shown above. Her name and DOB will remain the same throughout his/her education. So the student has a fixed value attribute that will never change in the future.
* These attributes are known as **stored attributes**.

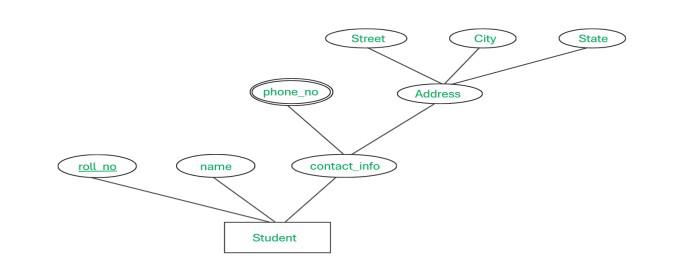


## Complex Attribute

* When multi-valued and composite attributes together form an attribute then it is called a **Complex attribute.**
* Complex attributes can have an unlimited number of sub-attributes.

### Example

* Here for the student, we created an attribute named contact\_info which further decomposed into phone\_no + Address.
* The address is a composite attribute which is further divided into simple attributes and phone\_no is a multivalued attribute.
* This indicates that the contact\_info attribute is made from the multi-valued and composite attribute.



**Tuple**

A tuple in a database management system is one record in the context of relational databases (one row). You can compare the data present in the database with a spreadsheet, with rows (known as tuples) and columns (known as fields or attributes) representing various data types.

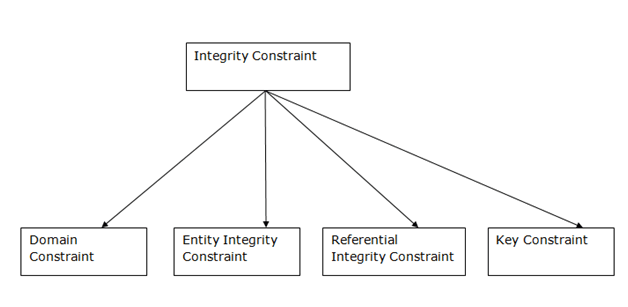
## ****Importance of NULL Value****

* It is important to understand that a NULL value differs from a zero value.
* A NULL value is used to represent a missing value, but it usually has one of three different interpretations:
  + The value unknown (value exists but is not known)
  + Value not available (exists but is purposely withheld)
  + Attribute not applicable (undefined for this tuple)
* It is often not possible to determine which of the meanings is intended. Hence, SQL does not distinguish between the different meanings of NULL.

# Constraints on Relational Database Model

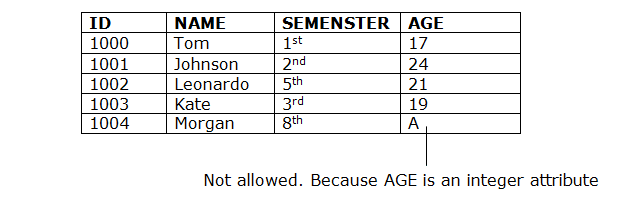
* Integrity constraints are a set of rules. It is used to maintain the quality of information.
* Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
* Thus, integrity constraint is used to guard against accidental damage to the database.

## Types of Integrity Constraint



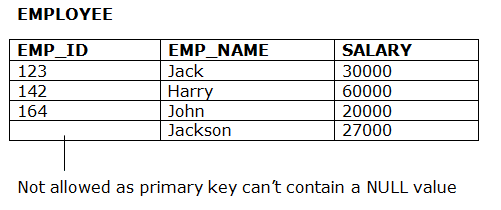
### 1. Domain constraints

* Domain constraints can be defined as the definition of a valid set of values for an
* attribute.
* The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

**Example:**

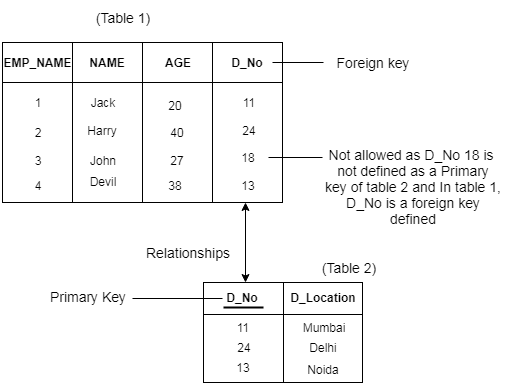
### 2. Entity integrity constraints

* The entity integrity constraint states that primary key value can't be null.
* This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
* A table can contain a null value other than the primary key field.

**Example:**  


### 3. Referential Integrity Constraints

* A referential integrity constraint is specified between two tables.
* In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

**Example:**  


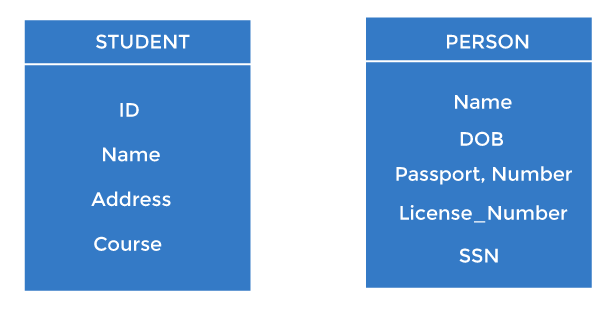
### 4. Key constraints

* Keys are the entity set that is used to identify an entity within its entity set uniquely.
* An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

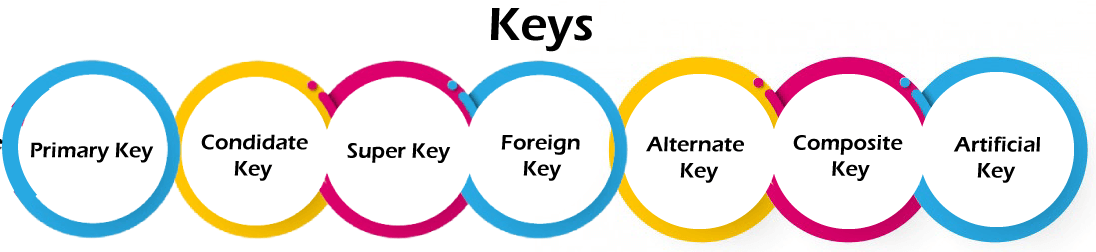
# Keys

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

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

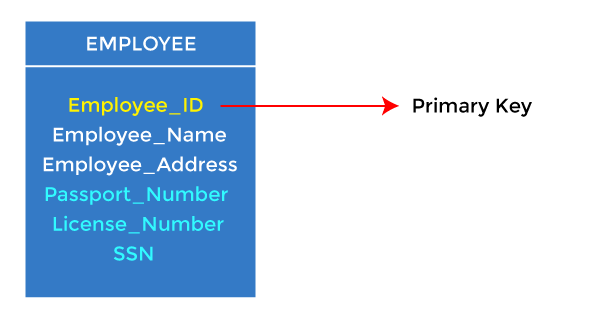


## Types of keys:



### 1. Primary key

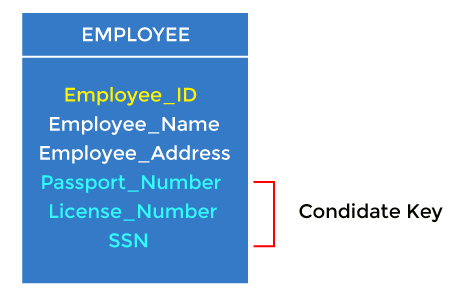
* It is the first key used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.
* In the EMPLOYEE table, ID can be the primary key since it is unique for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



### 2. Candidate key

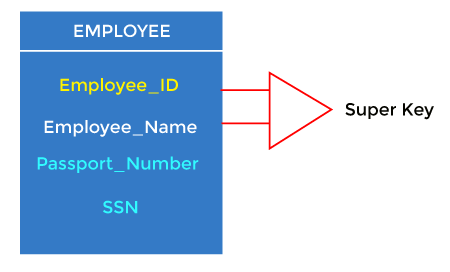
* A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
* Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

**For example:** In the EMPLOYEE table, id is best suited for the primary key. The rest of the attributes, like SSN, Passport\_Number, License\_Number, etc., are considered a candidate key.



### 3. Super Key

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

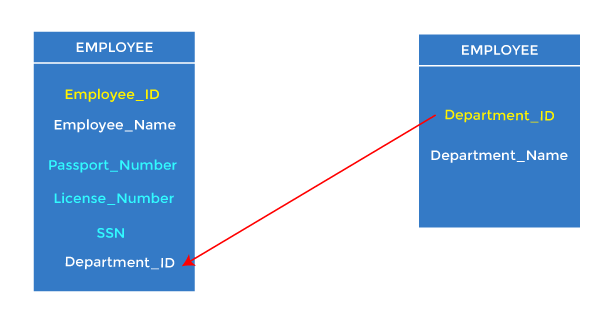


**For example:** In the above EMPLOYEE table, for(EMPLOEE\_ID, EMPLOYEE\_NAME), the name of two employees can be the same, but their EMPLYEE\_ID can't be the same. Hence, this combination can also be a key.

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

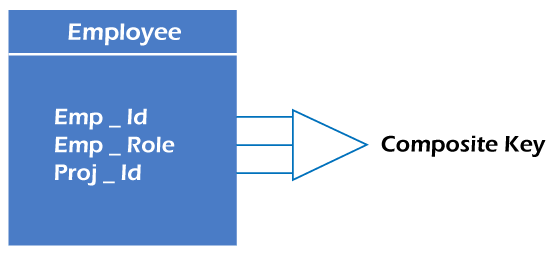
### 4. Foreign key

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



### 5. Composite key

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



## **For example,** in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously. So the primary key will be composed of all three attributes, namely Emp\_ID, Emp\_role, and Proj\_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute

## **Advantages of Relational Database Model**

* It is simpler than the [hierarchical model](https://www.geeksforgeeks.org/hierarchical-model-in-dbms/) and [network model](https://www.geeksforgeeks.org/network-model-in-dbms/).
* It is easy and simple to understand.
* Its structure can be changed anytime upon requirement.
* ****Data Integrity**:** The relational database model enforces data integrity through various constraints such as primary keys, foreign keys, and unique constraints. This ensures that the data in the database is accurate, consistent, and valid.
* ****Flexibility:****The relational database model is highly flexible and can handle a wide range of data types and structures. It also allows for easy modification and updating of the data without affecting other parts of the database.
* ****Scalability:****The relational database model can scale to handle large amounts of data by adding more tables, indexes, or partitions to the database. This allows for better performance and faster query response times.
* ****Security:****The relational database model provides robust security features to protect the data in the database. These include user authentication, authorization, and encryption of sensitive data.
* ****Data consistency:****The relational database model ensures that the data in the database is consistent across all tables. This means that if a change is made to one table, the corresponding changes will be made to all related tables.
* ****Query Optimization:**** The relational database model provides a query optimizer that can analyze and optimize SQL queries to improve their performance. This allows for faster query response times and better scalability.

## Disadvantages of the Relational Model

* Few database relations have certain limits which can’t be expanded further.
* It can be complex and it becomes hard to use.
* **Complexity:**The relational model can be complex and difficult to understand, particularly for users who are not familiar with SQL and database design principles. This can make it challenging to set up and maintain a relational database.
* **Performance:** The relational model can suffer from performance issues when dealing with large data sets or complex queries. In particular, joins between tables can be slow, and indexing strategies can be difficult to optimize.
* **Scalability:**While the relational model is generally scalable, it can become difficult to manage as the database grows in size. Adding new tables or indexes can be time-consuming, and managing relationships between tables can become complex.
* **Cost:**Relational databases can be expensive to license and maintain, particularly for large-scale deployments. Additionally, relational databases often require dedicated hardware and specialized software to run, which can add to the cost.
* **Limited flexibility:** The relational model is designed to work with tables that have predefined structures and relationships. This can make it difficult to work with data that does not fit neatly into a table-based format, such as unstructured or semi-structured data.
* **Data redundancy:** In some cases, the relational model can lead to data redundancy, where the same data is stored in multiple tables. This can lead to inefficiencies and can make it difficult to ensure data consistency across the database.

**Relational Algebra**

Relational Algebra is a procedural query language. The main purpose of using Relational Algebra is to define operators that transform one or more input relations into an output relation. Given that these operators accept relations as input and produce relations as output, they can be combined and used to express potentially complex queries that transform potentially many input relations (whose data are stored in the database) into a single output relation (the query results).

## Fundamental Operators

These are the [basic/fundamental operators](https://www.geeksforgeeks.org/basic-operators-in-relational-algebra-2/) used in Relational Algebra.

1. [Selection(σ)](https://www.geeksforgeeks.org/select-operation-in-relational-algebra/)
2. [Projection(π)](https://www.geeksforgeeks.org/difference-between-selection-and-projection-in-dbms/)
3. [Union(U)](https://www.geeksforgeeks.org/sql-union-operator/)
4. [Set Difference(-)](https://www.geeksforgeeks.org/set-theory-operations-in-relational-algebra/)
5. [Set Intersection(∩)](https://www.geeksforgeeks.org/sql-intersect-clause/)
6. [Rename(ρ)](https://www.geeksforgeeks.org/rename-operation-in-relational-algebra/)
7. [Cartesian Product(X)](https://www.geeksforgeeks.org/cartesian-product-operation-in-relational-algebra/)

**1. Selection(σ):**It is used to select required tuples of the [relations](https://www.geeksforgeeks.org/relation-in-maths/" \t "_self).

**Example:**

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

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

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

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

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

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

π(B,C)R will show following columns.

| **B** | **C** |
| --- | --- |
| 2 | 4 |
| 2 | 3 |
| 3 | 4 |

**Note:**By Default, projection removes duplicate data.

**3. Union(U):** Union operation in relational algebra is the same as union operation in [set theory](https://www.geeksforgeeks.org/set-operations/).

**Example:**

**FRENCH**

| **Student\_Name** | **Roll\_Number** |
| --- | --- |
| Ram | 01 |
| Mohan | 02 |
| Vivek | 13 |
| Geeta | 17 |

**GERMAN**

| **Student\_Name** | **Roll\_Number** |
| --- | --- |
| Vivek | 13 |
| Geeta | 17 |
| Shyam | 21 |
| Rohan | 25 |

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

π(Student\_Name)FRENCH U π(Student\_Name)GERMAN

| **Student\_Name** |
| --- |
| Ram |
| Mohan |
| Vivek |
| Geeta |
| Shyam |
| Rohan |

**Note:** The only constraint in the union of two relations is that both relations must have the same set of Attributes.

**4. Set Difference(-):** Set Difference in relational algebra is the same set difference operation as in set theory.

**Example:**From the above table of FRENCH and GERMAN, Set Difference is used as follows

π(Student\_Name)FRENCH - π(Student\_Name)GERMAN

| **Student\_Name** |
| --- |
| Ram |
| Mohan |

**Note:** The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

**5. Set Intersection(∩):**Set Intersection in relational algebra is the same set intersection operation in set theory.

**Example:**From the above table of FRENCH and GERMAN, the Set Intersection is used as follows

π(Student\_Name)FRENCH ∩ π(Student\_Name)GERMAN

| **Student\_Name** |
| --- |
| Vivek |
| Geeta |

**Note:** The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

**6. Rename(ρ):** Rename is a unary operation used for renaming attributes of a relation.

**ρ(a/b)R** will rename the attribute 'b' of the relation by 'a'.

**7. Cross Product(X):** Cross-product between two relations. Let’s say A and B, so the cross product between A X B will result in all the attributes of A followed by each attribute of B. Each record of A will pair with every record of B.

**Example:**

**A**

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

**B**

| **ID** | **Course** |
| --- | --- |
| 1 | DS |
| 2 | DBMS |

**A X B**

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

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

## Derived Operators

These are some of the [derived operators](https://www.geeksforgeeks.org/extended-operators-in-relational-algebra/), which are derived from the fundamental operators.

1. [Natural Join(⋈)](https://www.geeksforgeeks.org/sql-natural-join/)
2. [Conditional Join](https://www.geeksforgeeks.org/extended-operators-in-relational-algebra/)

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

**Example:**

**EMP**

| **Name** | **ID** | **Dept\_Name** |
| --- | --- | --- |
| A | 120 | IT |
| B | 125 | HR |
| C | 110 | Sales |
| D | 111 | IT |

**DEPT**

| **Dept\_Name** | **Manager** |
| --- | --- |
| Sales | Y |
| Production | Z |
| IT | A |

Natural join between EMP and DEPT with condition :

**EMP.Dept\_Name = DEPT.Dept\_Name**

**EMP ⋈ DEPT**

| **Name** | **ID** | **Dept\_Name** | **Manager** |
| --- | --- | --- | --- |
| A | 120 | IT | A |
| C | 110 | Sales | Y |
| D | 111 | IT | A |

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

**Example:**

**R**

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

**S**

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

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

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

## Relational Calculus

As Relational Algebra is a procedural query language, [Relational Calculus](https://www.geeksforgeeks.org/difference-between-relational-algebra-and-relational-calculus/) is a non-procedural query language. It basically deals with the end results. It always tells me what to do but never tells me how to do it.

There are two types of Relational Calculus

1. [Tuple Relational Calculus(TRC)](https://www.geeksforgeeks.org/tuple-relational-calculus-trc-in-dbms/)
2. [Domain Relational Calculus(DRC)](https://www.geeksforgeeks.org/domain-relational-calculus-in-dbms/)

**1. 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.

## 2. 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 |

**Basic SQL**

## Database Schema

* A database schema is a **logical representation of data** that shows how the data in a database should be stored logically. It shows how the data is organized and the relationship between the tables.
* Database schema contains table, field, views and relation between different keys like [primary key](https://www.geeksforgeeks.org/primary-key-in-dbms/), [foreign key](https://www.geeksforgeeks.org/foreign-key-constraint-in-sql/).

· Database schema provides the organization of data and the relationship between the stored data.

· Database schema defines a set of guidelines that control the database along with that it provides information about the way of accessing and modifying the data.

## Types of Database Schemas

**There are 3 types of database schema:**

### ****Physical Database Schema****

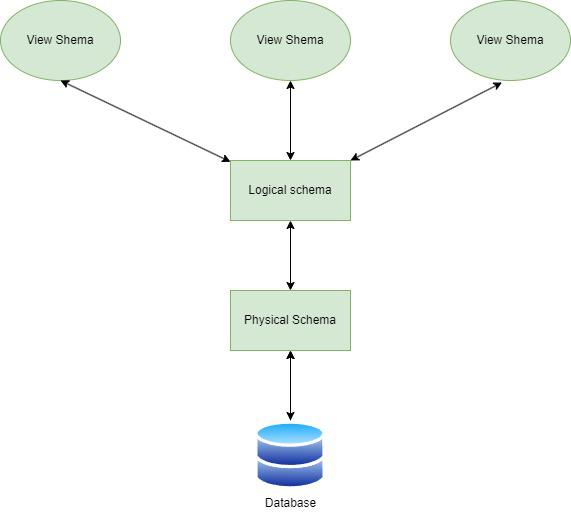
* A Physical schema defines, how the data or information is stored physically in the storage systems in the form of files & indices. This is the actual code or syntax needed to create the structure of a database, we can say that when we design a database at a physical level, it’s called physical schema.
* The Database administrator chooses where and how to store the data in the different blocks of storage.

### ****Logical Database Schema****

* A logical database schema defines all the logical constraints that need to be applied to the stored data, and also describes tables, views, entity relationships, and integrity constraints.
* The Logical schema describes how the data is stored in the form of tables & how the attributes of a table are connected.
* Using **ER modelling** the relationship between the components of the data is maintained.
* In logical schema different integrity constraints are defined in order to maintain the quality of insertion and update the data.

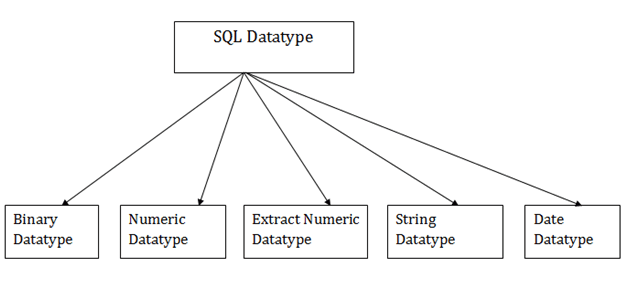
### ****View Database Schema****

* It is a view level design which is able to define the interaction between end-user and database.
* User is able to interact with the database with the help of the interface without knowing much about the stored mechanism of data in database.



## Data Types

An [SQL](https://www.geeksforgeeks.org/sql-tutorial/) developer must know what data type will be stored inside each column while creating a table. The data type guideline for SQL is to understand what type of data is expected inside each column and it also identifies how SQL will interact with the stored data.



### 1. Binary Datatypes

There are Three types of binary Datatypes which are given below:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| binary | It has a maximum length of 8000 bytes. It contains fixed-length binary data. |
| varbinary | It has a maximum length of 8000 bytes. It contains variable-length binary data. |
| image | It has a maximum length of 2,147,483,647 bytes. It contains variable-length binary data. |

### 2. Approximate Numeric Datatype :

The subtypes are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Data type** | **From** | **To** | **Description** |
| float | -1.79E + 308 | 1.79E + 308 | It is used to specify a floating-point value e.g. 6.2, 2.9 etc. |
| real | -3.40e + 38 | 3.40E + 38 | It specifies a single precision floating point number |

### 3. Exact Numeric Datatype

The subtypes are given below:

|  |  |
| --- | --- |
| **Data type** | **Description** |
| int | It is used to specify an integer value. |
| smallint | It is used to specify small integer value. |
| bit | It has the number of bits to store. |
| decimal | It specifies a numeric value that can have a decimal number. |
| numeric | It is used to specify a numeric value. |

### 4. Character String Datatype

The subtypes are given below:

|  |  |
| --- | --- |
| **Data type** | **Description** |
| char | It has a maximum length of 8000 characters. It contains Fixed-length non-unicode characters. |
| varchar | It has a maximum length of 8000 characters. It contains variable-length non-unicode characters. |
| text | It has a maximum length of 2,147,483,647 characters. It contains variable-length non-unicode characters. |

### 5. Date and time Data types

The sub types are given below:

|  |  |
| --- | --- |
| **Datatype** | **Description** |
| date | It is used to store the year, month, and days value. |
| time | It is used to store the hour, minute, and second values. |
| timestamp | It stores the year, month, day, hour, minute, and the second value. |

**Table Definitions:**

**Definition Language** actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

### List of DDL Commands:

Here are all the main DDL (Data Definition Language) commands along with their syntax:

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| [CREATE](https://www.geeksforgeeks.org/sql-create" \t "https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/_blank) | Create database or its objects (table, index, function, views, store procedure, and triggers) | **CREATE** TABLE table\_name (column1 data\_type, column2 data\_type, ...); |
| [DROP](https://www.geeksforgeeks.org/sql-drop-truncate" \t "https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/_blank) | Delete objects from the database | **DROP** TABLE table\_name; |
| [ALTER](https://www.geeksforgeeks.org/sql-alter-add-drop-modify" \t "https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/_blank) | Alter the structure of the database | **ALTER** TABLE table\_name ADD COLUMN column\_name data\_type; |

### CREATE Table

SQL create table is used to create a table in the database. To define the table, you should define the name of the table and also define its columns and column's data type.

**Syntax**

create table "table\_name"

( "column1" "data type",

column2" "data type",

column3" "data type",

"columnN" "data type"

);

**Example**

SQL> CREATE TABLE EMPLOYEE (

EMP\_ID INT                           NOT NULL,

EMP\_NAME VARCHAR (25) NOT NULL,

PHONE\_NO INT                         NOT NULL,

ADDRESS CHAR (30),

PRIMARY KEY (ID)

);

If you create the table successfully, you can verify the table by looking at the message by the SQL server. Else you can use DESC command as follows:

**SQL> DESC EMPLOYEE;**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| EMP\_ID | int(11) | NO | PRI | NULL |  |
| EMP\_NAME | varchar(25) | NO |  | NULL |  |
| PHONE\_NO | NO | int(11) |  | NULL |  |
| ADDRESS | YES |  |  | NULL | char(30) |

### DROP table

A SQL drop table is used to delete a table definition and all the data from a table. When this command is executed, all the information available in the table is lost forever, so you have to very careful while using this command.

**Syntax**

DROP TABLE "table\_name";

Example

SQL>DROP TABLE EMPLOYEE;

# ALTER TABLE

**SQL ALTER TABLE** command can add, delete, or modify columns of an existing table.

This article discusses the SQL ALTER TABLE statement with examples and syntax.

## ALTER TABLE STATEMENT

The **ALTER TABLE statement in SQL** is used to add, remove, or modify columns in an existing table. The ALTER TABLE statement is also used to add and remove various constraints on existing tables.

It allows for structural changes like adding new columns, modifying existing ones, deleting columns, and renaming columns within a table.

## Syntax

To alter/modify the table use the ALTER TABLE syntax:

**ALTER TABLE** table\_name  
**clause** [column\_name] [datatype];

Here, the clause is the operational clause of the ALTER TABLE statement. Some key clauses of the ALTER TABLE statement are:

### ADD – To add a new column to the table:

ALTER TABLE table\_name  
ADD column\_name datatype;

### MODIFY/ALTER – To change the data type of an existing column:

ALTER TABLE table\_name  
MODIFY COLUMN column\_name datatype;

### DROP – To delete an existing column from the table:

ALTER TABLE table\_name  
DROP COLUMN column\_name;

## SQL ALTER TABLE Examples

Below are the examples of ALTER TABLE statement. These examples demonstrates different use cases and shows **how to use ALTER TABLE** statement in SQL.

### SQL ALTER TABLE ADD Column Example

The following SQL query adds an “Email” column to the “Students” table:

**ALTER TABLE** Students  
**ADD** Email varchar(255);

### SQL ALTER TABLE DROP Column Example

The following query deletes the “Email” column from “Students” table:

**ALTER TABLE** Students  
**DROP COLUMN** Email;

### SQL ALTER TABLE MODIFY Column Example

**ALTER TABLE** table\_name  
**MODIFY COLUMN** column\_name datatype;

## SQL ALTER TABLE Queries

Suppose there is a student database:

| **ROLL\_NO** | **NAME** |
| --- | --- |
| 1 | Ram |
| 2 | Abhi |
| 3 | Rahul |
| 4 | Tanu |

To ADD 2 columns AGE and COURSE to table Student.

### Query:

ALTER TABLE Student ADD   
 (AGE number(3),COURSE varchar(40));

**Output:**

| **ROLL\_NO** | **NAME** | **AGE** | **COURSE** |
| --- | --- | --- | --- |
| 1 | Ram |  |  |
| 2 | Abhi |  |  |
| 3 | Rahul |  |  |
| 4 | Tanu |  |  |

MODIFY column COURSE in table Student.

### Query:

ALTER TABLE Student   
 MODIFY COURSE varchar(20);

After running the above query the maximum size of the Course Column is reduced to 20 from 40.

DROP column COURSE in table Student.

### Query:

ALTER TABLE Student   
DROP COLUMN COURSE;

**Output:**

| **ROLL\_NO** | **NAME** | **AGE** |
| --- | --- | --- |
| 1 | Ram |  |
| 2 | Abhi |  |
| 3 | Rahul |  |
| 4 | Tanu |  |

### DML commands

Here are all the main DML (Data Manipulation Language) commands along with their syntax:

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| [INSERT](https://www.geeksforgeeks.org/sql-insert-statement" \t "https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/_blank) | Insert data into a table | **INSERT** INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...); |
| [UPDATE](https://www.geeksforgeeks.org/sql-update-statement" \t "https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/_blank) | Update existing data within a table | **UPDATE** table\_name SET column1 = value1, column2 = value2 WHERE condition; |
| [DELETE](https://www.geeksforgeeks.org/sql-delete-statement" \t "https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/_blank) | Delete records from a database table | **DELETE** FROM table\_name WHERE condition; |

## SQL INSERT INTO Statement

* The **INSERT INTO** statement in [SQL](https://www.geeksforgeeks.org/sql-tutorial/" \t "https://www.geeksforgeeks.org/sql-insert-statement/_blank) is used to add new rows of data to a table in a database.
* There are two main ways to use the INSERT INTO statement by specifying the columns and values explicitly or by inserting values for all columns without specifying them.

## Syntax

There are two primary syntaxes of **INSERT INTO** statements depending on the requirements. The two syntaxes are:

### ****1. Only Values****

The first method is to specify only the value of data to be inserted without the column names.

**INSERT INTO** table\_name   
**VALUES** (value1, value2, value);

**Parameters:**

* **table\_name:** name of the table.
* **value1, value2:** value of first column, second column,… for the new record

### ****2. Column Names And Values Both****

In the second method we will specify both the columns which we want to fill and their corresponding values as shown below:

**INSERT INTO** table\_name (column1, column2, column3)   
**VALUES** ( value1, value2, value);

**Parameters:**

* **table\_name:** name of the table.
* **column1, column2..:** name of first column, second column.
* **value1, value2, value..:**  the values for each specified column of the new record.

## Examples of SQL INSERT INTO

For better understanding, let’s look at the SQL Server INSERT statement with examples.

Let us first create a table named ‘S**tudent**‘.

CREATE DATABASE StudentDB;

USE StudentDB;

CREATE TABLE Student ( ROLL\_NO INT PRIMARY KEY, NAME VARCHAR(50), ADDRESS VARCHAR(100), PHONE VARCHAR(15), AGE INT

);

INSERT INTO Student (ROLL\_NO, NAME, ADDRESS, PHONE, AGE) VALUES

(1, 'Ram', 'Delhi', 'XXXXXXXXXX', 18),

(2, 'Ramesh', 'Gurgaon', 'XXXXXXXXXX', 18),

(3, 'Sujit', 'Rohtak', 'XXXXXXXXXX', 20),

(4, 'Suresh', 'Rohtak', 'XXXXXXXXXX', 18);

**Created Table:**

| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| --- | --- | --- | --- | --- |
| 1 | Ram | Delhi | xxxxxxxxxxxxxx | 18 |
| 2 | RAMESH | GURGAON | xxxxxxxxxxxxxx | 18 |
| 3 | SUJIT | ROHTAK | xxxxxxxxxxxxxx | 20 |
| 4 | SURESH | ROHTAK | xxxxxxxxxxxxxx | 18 |
| 3 | SUJIT | ROHTAK | xxxxxxxxxxxxxx | 20 |
| 2 | RAMESH | GURGAON | xxxxxxxxxxxxxx | 18 |

### ****Example 2: Insert Values to Specified Columns Using INSERT INTO Example****

If we want to insert values in the specified columns then we use the following query.

**Query:**

**INSERT INTO** Student (ROLL\_NO, NAME, Age)   
**VALUES** ('5','PRATIK','19');

**Output:**

The table **Student** will now look like this:

| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **Age** |
| --- | --- | --- | --- | --- |
| 1 | Ram | Delhi | XXXXXXXXXX | 18 |
| 2 | RAMESH | GURGAON | XXXXXXXXXX | 18 |
| 3 | SUJIT | ROHTAK | XXXXXXXXXX | 20 |
| 4 | SURESH | Delhi | XXXXXXXXXX | 18 |
| 3 | SUJIT | ROHTAK | XXXXXXXXXX | 20 |
| 2 | RAMESH | GURGAON | XXXXXXXXXX | 18 |
| 5 | PRATIK | null | null | 19 |

## UPDATE Statement in SQL

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.

## ****Update Syntax****

The syntax for SQL UPDATE Statement is :

UPDATE table\_name SET column1 = value1, column2 = value2,…   
WHERE condition;

Where,

* **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

### ****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.

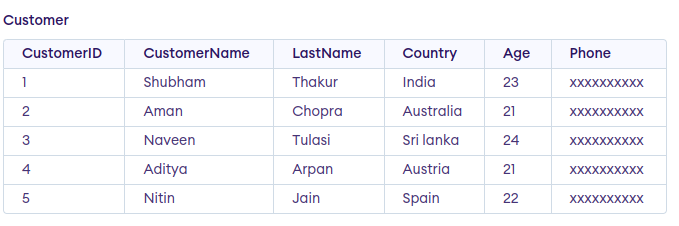
### ****Update Single Column**** Using UPDATE Statement Example

Update the column NAME and set the value to ‘Nitin’ in the rows where the Age is 22.

**Query:**

**UPDATE** Customer **SET** CustomerName   
= 'Nitin' WHERE Age = 22;

**Output:**



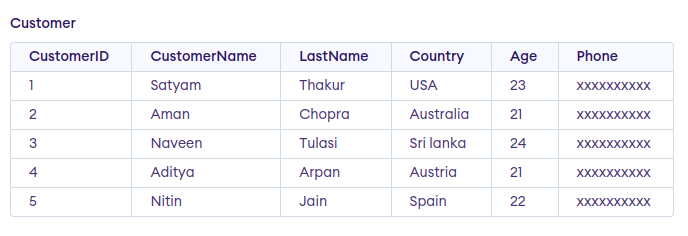
### ****Updating Multiple Columns using UPDATE Statement Example****

Update the columns NAME to ‘Satyam’ and Country to ‘USA’ where CustomerID is 1.

**Query:**

**UPDATE** Customer **SET** CustomerName = 'Satyam',   
Country = 'USA' **WHERE** CustomerID = 1;

**Output**:



**Note:** For updating multiple columns we have used comma(,) to separate the names and values of two columns.

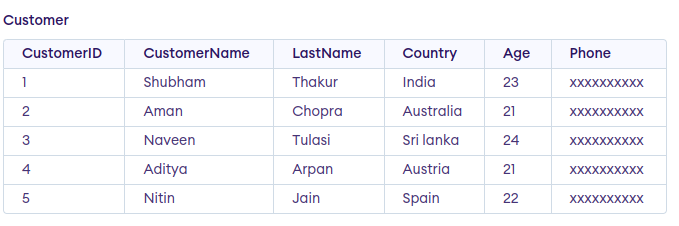
### ****Update Single Column**** Using UPDATE Statement Example

Update the column NAME and set the value to ‘Nitin’ in the rows where the Age is 22.

**Query:**

**UPDATE** Customer **SET** CustomerName   
= 'Nitin' WHERE Age = 22;

**Output:**



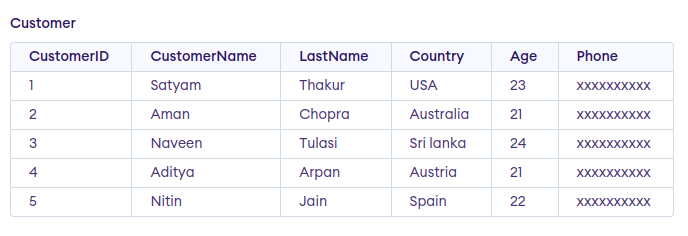
### ****Updating Multiple Columns using UPDATE Statement Example****

Update the columns NAME to ‘Satyam’ and Country to ‘USA’ where CustomerID is 1.

**Query:**

**UPDATE** Customer **SET** CustomerName = 'Satyam',   
Country = 'USA' **WHERE** CustomerID = 1;

**Output**:



**Note:** For updating multiple columns we have used comma(,) to separate the names and values of two columns.

# DELETE Statement

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

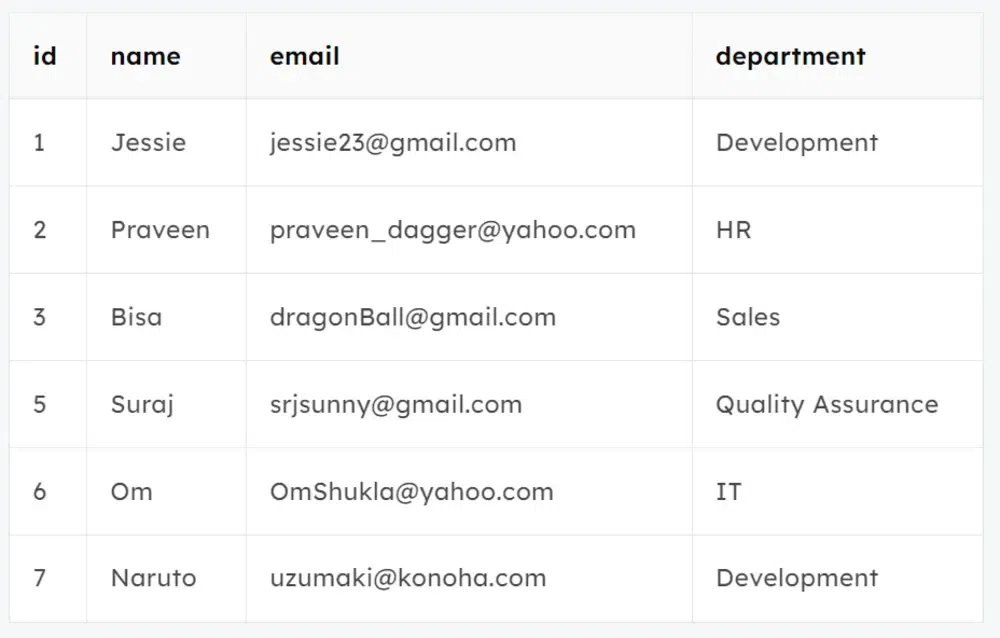
## ****Deleting Single Record****

You can delete the records named Rithvik by using the below query:

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

DELETE FROM GFG\_Employees WHERE NAME = 'Rithvik';

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



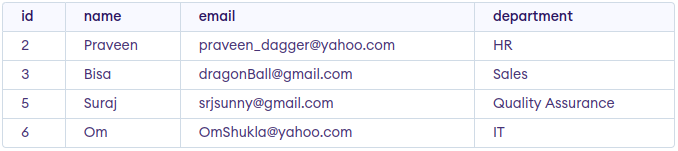
## ****Deleting Multiple Records****

Delete the rows from the table  GFG\_Employees where the department is “Development”. This will delete 2 rows(the first row and the seventh row).

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

DELETE FROM GFG\_Employees   
WHERE department = 'Development';

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



output

## ****Delete All of the Records****

To remove all the entries from the table, you can use the following query:

****Query****

DELETE FROM GFG\_EMPLOyees;  
Or  
DELETE \* FROM GFG\_EMPLOyees;

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

All of the records in the table will be deleted, there are no records left to display. The table GFG\_EMPLOyees  will become empty.

