In DBMS (Database Management Systems), constraints are guidelines or limitations imposed on database tables to maintain the integrity, correctness, and consistency of the data. Constraints can be used to enforce data linkages across tables, verify that data is unique, and stop the insertion of erroneous data. A database needs constraints to be reliable and of high quality.

## What are the Constraints of DBMS?

In DBMS, constraints refer to limitations placed on data or data processes. This indicates that only a particular type of data may be entered into the database or that only a particular sort of operation can be performed on the data inside.

Constraints thereby guarantee data accuracy in a database management system (DBMS).

**The following can be guaranteed via constraints**

**Data Accuracy** − Data accuracy is guaranteed by constraints, which make sure that only true data is entered into a database. For example, a limitation may stop a user from entering a negative value into a field that only accepts positive numbers.

**Data Consistency** − The consistency of data in a database can be upheld by using constraints. These constraints are able to ensure that the primary key value in one table is followed by the foreign key value in another table.

**Data integrity** − The accuracy and completeness of the data in a database are ensured by constraints. For example, a constraint can stop a user from putting a null value into a field that requires one.

### **Types of Constraints in DBMS**

* Domain Constraints
* Key Constraints
* Entity Integrity Constraints
* Referential Integrity Constraints
* Tuple Uniqueness Constraints

## Domain Constraints

In a database table, domain constraints are guidelines that specify the acceptable values for a certain property or field. These restrictions guarantee data consistency and aid in preventing the entry of inaccurate or inconsistent data into the database. The following are some instances of domain restrictions in a DBMS −

* **Data type constraints** − These limitations define the kinds of data that can be kept in a column. A column created as VARCHAR can take string values, but a column specified as INTEGER can only accept integer values.
* **Length Constraints** − These limitations define the largest amount of data that may be put in a column. For instance, a column with the definition VARCHAR(10) may only take strings that are up to 10 characters long.
* **Range constraints** − The allowed range of values for a column is specified by range restrictions. A column designated as DECIMAL(5,2), for example, may only take decimal values up to 5 digits long, including 2 decimal places.
* **Nullability constraints** − Constraints on a column's capacity to accept NULL values are known as nullability constraints. For instance, a column that has the NOT NULL definition cannot take NULL values.
* **Unique constraints** − Constraints that require the presence of unique values in a column or group of columns are known as unique constraints. For instance, duplicate values are not allowed in a column with the UNIQUE definition.
* **Check constraints** − Constraints for checking data: These constraints outline a requirement that must hold for any data placed into the column. For instance, a column with the definition CHECK (age > 0) can only accept ages that are greater than zero.
* **Default constraints** − Constraints by default: Default constraints automatically assign a value to a column in case no value is provided. For example, a column with a DEFAULT value of 0 will have 0 as its value if no other value is specified.

## Key Constraints

Key constraints are regulations that a DBMS uses to ensure data accuracy and consistency in a database. They define how the values in a table's one or more columns are related to the values in other tables, making sure that the data remains correct.

In DBMS, there are several key constraint kinds, including −

* **Primary Key Constraint** − A primary key constraint is an individual identifier for each record in a database. It guarantees that each database entry contains a single, distinct value—or a pair of values—that cannot be null—as its method of identification.
* **Foreign Key Constraint** − Reference to the primary key in another table is a foreign key constraint. It ensures that the values of a column or set of columns in one table correspond to the primary key column(s) in another table.
* **Unique Constraint** − In a database, a unique constraint ensures that no two values inside a column or collection of columns are the same.

## Entity Integrity Constraints

A database management system uses entity integrity constraints (EICs) to enforce rules that guarantee a table's primary key is unique and not null. The consistency and integrity of the data in a database are maintained by EICs, which are created to stop the formation of duplicate or incomplete entries.

Each item in a table in a relational database is uniquely identified by one or more fields known as the primary key. EICs make a guarantee that every row's primary key value is distinct and not null. Take the "Employees" table, for instance, which has the columns "EmployeeID" and "Name." The table's primary key is the EmployeeID column. An EIC on this table would make sure that each row's unique EmployeeID value is there and that it is not null.

If you try to insert an entry with a duplicate or null EmployeeID, the database management system will reject the insertion and produce an error. This guarantees that the information in the table is correct and consistent.

EICs are a crucial component of database architecture and assist guarantee the accuracy and dependability of the data contained in a database.

## Referential Integrity Constraints

A database management system will apply referential integrity constraints (RICs) in order to preserve the consistency and integrity of connections between tables. By preventing links between entries that don't exist from being created or by removing records that have related records in other tables, RICs guarantee that the data in a database is always consistent.

By the use of foreign keys, linkages between tables are created in relational databases. A column or collection of columns in one table that is used as a foreign key to access the primary key of another table. RICs make sure there are no referential errors and that these relationships are legitimate.

Consider the "Orders" and "Customers" tables as an illustration. The primary key column in the "Customers" database corresponds to the foreign key field "CustomerID" in the "Orders" dataset. A RIC on this connection requires that each value in the "CustomerID" column of the "Orders" database exist in the "Customers" table's primary key column.

If an attempt was made to insert a record into the "Orders" table with a non-existent "CustomerID" value, the database management system would reject the insertion and notify the user of an error.

Similar to this, the database management system would either prohibit the deletion or cascade the deletion in order to ensure referential integrity if a record in the "Customers" table was removed and linked entries in the "Orders" table.

In general, RICs are a crucial component of database architecture and assist guarantee that the information contained in a database is correct and consistent throughout time.

## Tuple Uniqeness Contraints

A database management system uses constraints called tuple uniqueness constraints (TUCs) to make sure that every entry or tuple in a table is distinct. TUCs impose uniqueness on the whole row or tuple, in contrast to Entity Integrity Constraints (EICs), which only enforce uniqueness on certain columns or groups of columns.

TUCs, then, make sure that no two rows in a table have the same values for every column. Even if the individual column values are not unique, this can be helpful in cases when it is vital to avoid the production of duplicate entries.

Consider the "Sales" table, for instance, which has the columns "TransactionID," "Date," "CustomerID," and "Amount." Even if individual column values could be duplicated, a TUC on this table would make sure that no two rows have the same values in all four columns.

The database management system would reject the insertion and generate an error if an attempt was made to enter a row with identical values in each of the four columns as an existing entry. This guarantees the uniqueness and accuracy of the data in the table.

TUCs may be a helpful tool for ensuring data correctness and consistency overall, especially when it's vital to avoid the generation of duplicate entries.

## Conclusion

Constraints are a crucial part of every database management system, and creating and maintaining high-quality databases requires a grasp of how to apply them effectively. To guarantee data's correctness, consistency, and integrity, constraints in DBMS apply rules to the data. They stop data from being added, altered, or removed that is incorrect or incomplete. EICs, RICs, TUCs, and Check Constraints are a few different kinds of constraints. Building and maintaining high-quality databases that allow for informed business decisions requires constraints.

# SQL Constraints

In a database table, we can add rules to a column known as **constraints**. These rules control the data that can be stored in a column.

For example, if a column has NOT NULL constraint, it means the column cannot store NULL values.

The constraints used in SQL are:

|  |  |
| --- | --- |
| Constraint | Description |
| NOT NULL | values cannot be null |
| UNIQUE | values cannot match any older value |
| PRIMARY KEY | used to uniquely identify a row |
| FOREIGN KEY | references a row in another table |
| CHECK | validates condition for new value |
| DEFAULT | set default value if not passed |
| CREATE INDEX | used to speedup the read process |

**Note:** These constraints are also called integrity constraints.

## NOT NULL Constraint

The NOT NULL constraint in a column means that the column cannot store NULL values. For example,

CREATE TABLE Colleges (

college\_id INT NOT NULL,

college\_code VARCHAR(20) NOT NULL,

college\_name VARCHAR(50)

);

[Run Code](https://www.programiz.com/sql/online-compiler)

Here, the college\_id and the college\_code columns of the Colleges table won't allow NULL values.

To learn more, visit [SQL NOT NULL Constraint](https://www.programiz.com/sql/not-null).

## UNIQUE Constraint

The UNIQUE constraint in a column means that the column must have unique value. For example,

CREATE TABLE Colleges (

college\_id INT NOT NULL UNIQUE,

college\_code VARCHAR(20) UNIQUE,

college\_name VARCHAR(50)

);

[Run Code](https://www.programiz.com/sql/online-compiler)

Here, the value of the college\_code column must be unique. Similarly, the value of college\_id must be unique as well as it cannot store NULL values.

To learn more, visit [SQL UNIQUE Constraint](https://www.programiz.com/sql/unique).

## PRIMARY KEY Constraint

The PRIMARY KEY constraint is simply a combination of NOT NULL and UNIQUE constraints. It means that the column value is used to uniquely identify the row. For example,

CREATE TABLE Colleges (

college\_id INT PRIMARY KEY,

college\_code VARCHAR(20) NOT NULL,

college\_name VARCHAR(50)

);

[Run Code](https://www.programiz.com/sql/online-compiler)

Here, the value of the college\_id column is a unique identifier for a row. Similarly, it cannot store NULL value and must be UNIQUE.

To learn more, visit [SQL PRIMARY KEY](https://www.programiz.com/sql/primary-key).

## FOREIGN KEY Constraint

The FOREIGN KEY (REFERENCES in some databases) constraint in a column is used to reference a record that exists in another table. For example,

CREATE TABLE Orders (

order\_id INT PRIMARY KEY,

customer\_id int REFERENCES Customers(id)

);

[Run Code](https://www.programiz.com/sql/online-compiler)

Here, the value of the college\_code column references the row in another table named Customers.

It means that the value of customer\_id in the Orders table must be a value from the id column of the Customers table.

To learn more, visit [SQL FOREIGN KEY](https://www.programiz.com/sql/foreign-key).

## CHECK Constraint

The CHECK constraint checks the condition before allowing values in a table. For example,

CREATE TABLE Orders (

order\_id INT PRIMARY KEY,

amount int CHECK (amount >= 100)

);

[Run Code](https://www.programiz.com/sql/online-compiler)

Here, the value of the amount column must be **greater than or equal to 100**. If not, the SQL statement results in an error.

To learn more, visit [SQL CHECK Constraint](https://www.programiz.com/sql/check).

## DEFAULT Constraint

The DEFAULT constraint is used to set the default value if we try to store NULL in a column. For example,

CREATE TABLE College (

college\_id INT PRIMARY KEY,

college\_code VARCHAR(20),

college\_country VARCHAR(20) DEFAULT 'US'

);

[Run Code](https://www.programiz.com/sql/online-compiler)

Here, the default value of the college\_country column is **US**.

If we try to store the NULL value in the college\_country column, its value will be **US**.

To learn more, visit [SQL DEFAULT Constraint](https://www.programiz.com/sql/default).

## CREATE INDEX Constraint

If a column has CREATE INDEX constraint, it's faster to retrieve data if we use that column for data retrieval. For example,

-- create table

CREATE TABLE Colleges (

college\_id INT PRIMARY KEY,

college\_code VARCHAR(20) NOT NULL,

college\_name VARCHAR(50)

);

-- create index

CREATE INDEX college\_index

ON Colleges(college\_code);

[Run Code](https://www.programiz.com/sql/online-compiler)

Here, the SQL command creates an index named customers\_index on the Customers table using customer\_id column.

**Note:** We cannot see the speed difference with less records in a table. However, we can easily notice the speed difference between using indexes and not using indexes.

To learn more, visit [SQL CREATE INDEX](https://www.programiz.com/sql/create-index).