Key Constraints in DBMS (Database Management System)

Key constraints are rules in a database that ensure the integrity and uniqueness of data stored in a database. These constraints are essential for maintaining data consistency, accuracy, and validity. The most common key constraints in DBMS are:

1. Primary Key Constraint

The **primary key** constraint ensures that each record in a table is uniquely identifiable. A primary key is a column or a combination of columns that uniquely identify each row in a table. No two rows can have the same primary key value.

- **Rules**:
    - The primary key column cannot contain `NULL` values.
    - All values in the primary key column must be unique.
- **Example**: Suppose we have a table called `Students`:

StudentID (Primary Key)	Name	Age
101	John Smith	22
102	Alice Wong	20
103	Bob Lee	21

- In this case, `StudentID` is the primary key, ensuring that each student has a unique identifier.

---

2. Foreign Key Constraint

A **foreign key** is a column or a set of columns in one table that refers to the primary key of another table. The foreign key constraint ensures referential integrity, meaning that a record in the child table must match a record in the parent table.

- **Rules**:
    - The foreign key value must exist in the parent table or be `NULL` (if the foreign key allows null values).

    - The values in the foreign key column must match the values in the referenced primary key column of the parent table.
- **Example**: Consider two tables: `Orders` and `Customers`.

Customers Table:

CustomerID (Primary Key)	Name	Address
1	John Smith	123 Main St
2	Alice Wong	456 Oak St

-

Orders Table:

OrderID	OrderDate	CustomerID (Foreign Key)
101	2025-02-10	1
102	2025-02-11	2

- In this case, `CustomerID` in the `Orders` table is a foreign key that references the `CustomerID` in the `Customers` table. The foreign key constraint ensures that only existing customers can place orders.

---

3. Unique Constraint

The **unique** constraint ensures that all values in a column (or a combination of columns) are distinct, meaning that no two rows in the table can have the same value in that column (or combination of columns).

- **Rules**:
    - The column with a unique constraint must contain unique values.
    - Unlike the primary key, a unique column can contain `NULL` values (unless otherwise specified).
- **Example**: Consider a `Users` table:

UserID (Primary Key)	Username (Unique)	Email

1	jsmith	john@example.com
2	awong	alice@example.com
3	rdoe	robert@example.com

- In this case, the `Username` column has a unique constraint, ensuring that no two users have the same username.

---

4. Not Null Constraint

The **not null** constraint ensures that a column cannot have `NULL` values. This is used to enforce that every record must have a value for that column.

- **Rules**:
    - A column defined with the `NOT NULL` constraint cannot have a `NULL` value in any row.
- **Example**: Consider a `Products` table:

ProductID (Primary Key)	ProductName	Price
1	Laptop	1000
2	Smartphone	700
3	Tablet	400

- Here, the `ProductName` and `Price` columns can be defined with the `NOT NULL` constraint to ensure that every product must have a name and price.

---

5. Check Constraint

The **check** constraint is used to limit the range of values that can be entered into a column. It allows the definition of a condition or rule that must be satisfied for the data to be valid.

- **Rules**:
    - The condition defined by the check constraint must evaluate to `TRUE` for every row in the table.
- **Example**: Consider an `Employees` table where you want to ensure that the `Age` of an employee is between 18 and 65:

EmployeeID (Primary Key)	Name	Age
1	John Smith	22
2	Alice Wong	20

To enforce this, you would apply a check constraint:

ALTER TABLE Employees ADD CONSTRAINT AgeCheck CHECK (Age BETWEEN 18 AND 65);

- This ensures that only valid ages (between 18 and 65) are entered into the `Age` column.

---

6. Default Constraint

The **default** constraint provides a default value for a column when no value is specified during the insertion of a new row.

- **Rules**:
    - If no value is provided for a column with a default constraint, the default value is automatically inserted into that column.
- **Example**: Consider a `Products` table where the `Stock` column should have a default value of 0 if no value is provided:

ProductID	ProductName	Stock
1	Laptop	100
2	Smartphone	50

You can set a default value for `Stock`:

ALTER TABLE Products ADD CONSTRAINT DefaultStock DEFAULT 0 FOR Stock;

- Now, if a new product is inserted without specifying the stock quantity, the `Stock` column will default to 0.

---

7. Composite Key

A **composite key** is a key that consists of two or more columns used together to uniquely identify a row in a table. It is used when no single column is sufficient to uniquely identify a row.

- **Rules**:
    - All the columns in a composite key together must be unique.
    - The columns that make up the composite key can have `NULL` values (depending on the DBMS, but typically they should not).
- **Example**: Consider a `CourseEnrollments` table that stores student enrollments in courses, where no single column is unique enough to identify an enrollment:

StudentID	CourseID	EnrollmentDate
101	201	2025-02-10
102	202	2025-02-11

In this case, a **composite primary key** is created by combining `StudentID` and `CourseID`, ensuring that a student can only enroll in a course once:

ALTER TABLE CourseEnrollments
ADD CONSTRAINT CompositeKey PRIMARY KEY (StudentID, CourseID);

-

Conclusion

Key constraints are crucial for maintaining data integrity in a database. They help ensure uniqueness, accuracy, and consistency of the data stored. The most commonly used key constraints are **Primary Key**, **Foreign Key**, **Unique**, **Not Null**, **Check**, **Default**, and **Composite**

Key. By using these constraints, database designers can enforce rules and prevent invalid data from entering the system, thus ensuring the reliability and robustness of the database system.