

## MySQL, the types of keys :

- ☐ **Candidate Key**
- ☐ **Primary Key**
- ☐ **Foreign Key**
- ☐ **Unique Key**
- ☐ **Composite Key**

### 1.Candidate Key

A **candidate key** in MySQL (and relational databases in general) is a column or a set of columns that can uniquely identify a row in a table. These are the key attributes that have the potential to become the primary key but have not necessarily been chosen as the primary key. Here's a more detailed explanation:

#### Characteristics of a Candidate Key:

1. **Uniqueness:** Every value of the candidate key must be unique within the table. This ensures that no two rows have the same candidate key value.
2. **Minimality:** A candidate key must have the minimum number of columns necessary to maintain uniqueness. This means that removing any column from the candidate key would result in a loss of uniqueness.
3. **Non-nullability:** A candidate key must not contain **NULL** values because NULLs do not ensure uniqueness.

#### Example:

Consider a table employees with the following columns:

- employee\_id
- email
- phone\_number
- social\_security\_number

#### In this case:

employee\_id could be a **candidate key** because it uniquely identifies each employee.

email could be another candidate key if every employee has a unique email address.

social\_security\_number could be a candidate key if every employee has a unique social security number.

#### Difference Between Candidate Key and Primary Key:

- **Primary Key:** One of the candidate keys is chosen to be the primary key of the table. The primary key uniquely identifies a record and is used to establish relationships with other tables.
- **Candidate Key:** These are all the possible keys that can uniquely identify records. There can be multiple candidate keys, but only one is chosen as the primary key.

### Example in MySQL:

```
CREATE TABLE employees (  
    employee_id INT NOT NULL,  
    email VARCHAR(255) NOT NULL,  
    phone_number VARCHAR(20),  
    social_security_number VARCHAR(20) NOT NULL,  
    PRIMARY KEY (employee_id),  
    UNIQUE (email),  
    UNIQUE (social_security_number)  
);
```

### In this example:

- employee\_id, email, and social\_security\_number are candidate keys.
- employee\_id is chosen as the primary key.
- email and social\_security\_number are enforced as unique to maintain their candidacy.

### Why Use Candidate Keys:

- Candidate keys are useful because they ensure data integrity and help in normalizing the database. By identifying candidate keys, you can determine the best primary key and ensure that your data model supports unique identification of rows.

## 2. Primary Key

A **primary key** is a fundamental concept in database design used to uniquely identify each record in a table. Here are the key aspects and characteristics of a primary key:

### Characteristics of a Primary Key

1. **Uniqueness:**

- The primary key must contain unique values. No two rows in a table can have the same value for the primary key column(s). This ensures that each row is uniquely identifiable.

2. **Non-Nullable:**

- The primary key column(s) cannot contain NULL values. Each record must have a valid, non-null value for the primary key, which guarantees that every row can be uniquely identified.

3. **Single Column or Composite:**

A primary key can be defined using a single column (single-column primary key) or multiple columns (composite or compound primary key). The combination of the columns in a composite primary key must be unique for each row.

## Examples

### *Single-Column Primary Key*

```
CREATE TABLE employees (  
    employee_id INT AUTO_INCREMENT,  
    name VARCHAR(100),  
    PRIMARY KEY (employee_id)  
);
```

In this example, `employee_id` is a single-column primary key. Each employee must have a unique `employee_id`, and it cannot be NULL.

### Composite Primary Key

```
CREATE TABLE order_items (  
    order_id INT,  
    product_id INT,  
    quantity INT,  
    PRIMARY KEY (order_id, product_id)  
);
```

In this example, the primary key consists of two columns: `order_id` and `product_id`. The combination of these two columns must be unique for each row, ensuring that each order-product pair is unique in the table.

## 2. Foreign Key

The **FOREIGN KEY** constraint is used to prevent actions that would destroy links between tables.

A **FOREIGN KEY** is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the **child table**, and the table with the primary key is called the referenced or **parent table**.

Look at the following two tables:

**Employee**

user_id	Name	city
123	Sachin	Noida
124	Vikas	Delhi
125	Ajay	Pune

**User**

user_id	designation
123	Python
123	Mysql
125	PoweBI

Notice that the "user\_id" column in the "Employee" table points to the "user\_id" column in the "User" table.

The "User\_id" column in the "Employee" table is the PRIMARY KEY in the "Employee" table.

The "User\_id" column in the "User" table is a FOREIGN KEY in the "User" table.

The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

## Key Characteristics of a Foreign Key

1. **Establishes Relationships:**
  - A foreign key creates a relationship between two tables. It links a column (or a set of columns) in one table to the primary key (or a unique key) of another table.
2. **Enforces Referential Integrity:**
  - Ensures that the value in the foreign key column matches a value in the referenced column of the parent table or is NULL. This helps maintain consistency and accuracy of the data across related tables.
3. **Can Be NULL:**
  - Foreign key columns can contain NULL values if the relationship is optional. This means a record in the child table can exist without being linked to a record in the parent table.
4. **Cascading Actions:**
  - Foreign keys can be configured with cascading actions that automatically propagate changes from the parent table to the child table. Options include:
    - **ON DELETE CASCADE:** Automatically deletes related rows in the child table when a row in the parent table is deleted.
    - **ON UPDATE CASCADE:** Automatically updates related rows in the child table when the referenced key in the parent table is updated.
    - **ON DELETE SET NULL:** Sets the foreign key column to NULL in the child table when a row in the parent table is deleted.
    - **ON UPDATE SET NULL:** Sets the foreign key column to NULL in the child table when the referenced key in the parent table is updated.

## Example of Foreign Key Usage

### *Parent Table*

Consider a table `departments` where each department has a `unique_id` (Primary key):

```
CREATE TABLE departments (  
    department_id INT AUTO_INCREMENT,  
    department_name VARCHAR(100) NOT NULL,  
    PRIMARY KEY (department_id)  
);
```

### Child Table

Consider a table `employees` where each employee belongs to a department. The `department_id` column in the `employees` table is a foreign key that references the `department_id` in the `departments` table:

```
CREATE TABLE employees (  
    employee_id INT AUTO_INCREMENT,  
    first_name VARCHAR(50) NOT NULL,  
    last_name VARCHAR(50) NOT NULL,  
    department_id INT,  
    PRIMARY KEY (employee_id),  
    FOREIGN KEY (department_id) REFERENCES departments(department_id)  
);
```

## 3. Unique Key

A **unique key** (or unique constraint) is a database constraint used to ensure that all values in a column or a combination of columns are unique across all rows in a table. This means no two rows can have the same value(s) in the columns that are defined as unique keys.

### Key Characteristics of a Unique Key

1. **Uniqueness:**
  - Ensures that the values in the specified column(s) are unique across the table. This prevents duplicate values in the unique key column(s).
2. **Nullable Values:**
  - Unlike a primary key, a unique key column can contain NULL values. However, if multiple NULLs are allowed, they are considered distinct from each other, meaning multiple NULL values can exist in a column with a unique constraint.
3. **Multiple Unique Keys:**
  - A table can have multiple unique key constraints. Each unique key constraint ensures the uniqueness of the specified column(s) within the table.
4. **Index:**
  - MySQL automatically creates a unique index on the column(s) defined as unique keys, which helps improve query performance.
5. **Enforcement:**
  - The database system enforces the uniqueness constraint during insertions and updates to ensure that no duplicate values are added.

### Example of Unique Key Usage

### Table Definition with Unique Key

Consider a table `users` where you want to ensure that both the `email` and `username` columns contain unique values:

```
CREATE TABLE users (  
    user_id INT AUTO_INCREMENT,  
    username VARCHAR(50) UNIQUE,    -- Unique key on username  
    email VARCHAR(100) UNIQUE,     -- Unique key on email  
    password VARCHAR(255),  
    PRIMARY KEY (user_id)  
);
```

## 4.Composite Key

A **composite key** (or compound key) is a primary key that consists of two or more columns in a table. The combination of these columns is used to uniquely identify each record in the table. This type of key is used when a single column alone is insufficient to ensure the uniqueness of each row.

### Characteristics of a Composite Key

1. **Uniqueness:**
  - The combination of values in all columns included in the composite key must be unique across the table. This ensures that no two rows have the same combination of values in these columns.
2. **Multiple Columns:**
  - A composite key involves more than one column. Each column individually might not be unique, but the combination of all specified columns provides the uniqueness.
3. **Primary Key:**
  - A composite key can serve as the primary key for a table, providing a unique identifier for each row based on the combined values of multiple columns.
4. **Foreign Key:**
  - Composite keys can also be used as foreign keys in other tables, allowing complex relationships that span multiple columns.

### Example of a Composite Key

Consider a table `order_items` where each item in an order is uniquely identified by a combination of `order_id` and `product_id`:

### Table Definition with Composite Key

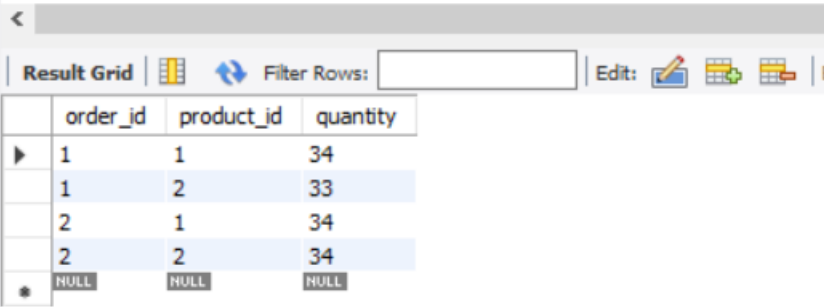
```
CREATE TABLE order_items (  
    order_id INT,  
    product_id INT,  
    quantity INT,  
    PRIMARY KEY (order_id, product_id)  
);
```

In this example:

- The primary key consists of two columns: `order_id` and `product_id`.
- The combination of `order_id` and `product_id` must be unique for each row. This means that each order can contain multiple products, but the combination of `order_id` and `product_id` must be unique across all rows.
- All possible combination as below.

In below example you can see all possible combination.

```
9 • INSERT into rohit(order_id,product_id,quantity)  
10 values  
11 (2,1,34);  
12  
13  
14 • select * from rohit;
```



	order_id	product_id	quantity
▶	1	1	34
	1	2	33
	2	1	34
	2	2	34
*	NULL	NULL	NULL

### Example of Composite Key in Relationships

Consider a junction table `student_courses` that links students to courses:



```
CREATE TABLE students (  
    student_id INT AUTO_INCREMENT,  
    name VARCHAR(100),  
    PRIMARY KEY (student_id)  
);  
  
CREATE TABLE courses (  
    course_id INT AUTO_INCREMENT,  
    course_name VARCHAR(100),  
    PRIMARY KEY (course_id)  
);  
  
CREATE TABLE student_courses (  
    student_id INT,  
    course_id INT,  
    enrollment_date DATE,  
    PRIMARY KEY (student_id, course_id),  
    FOREIGN KEY (student_id) REFERENCES students(student_id),  
    FOREIGN KEY (course_id) REFERENCES courses(course_id)  
);
```