## Module 9 - Keys in SQL

### What is a Key in SQL?

In SQL (Structured Query Language), a **key** is a column (or a group of columns) in a table that is used to **identify rows** in that table. Keys are like **labels or tags** that help the database know **which row is which**. They make sure that the data is **organized**, **unique**, and **connected** between tables.

Imagine a table like a list of names and phone numbers. If two people have the same name, how do we tell them apart? That's where keys come in.

**Why are keys important?**

1. **They make each row unique**
Keys make sure that every row in a table is different. This is important so we don't confuse or mix up data.
2. **They help link tables together**
In a real-world database, information is often stored in many tables. Keys help **connect** these tables so they can work together.
3.**They help search and update data quickly**
With keys, the database can find, sort, or update specific rows much faster.

### A simple example

Imagine a table called students:

| student_id | name | age |
|---|---|---|
| 101 | Alice | 20 |
| 102 | Bob | 21 |
| 103 | Alice | 22 |

Here, name is not unique—there are two Alices. But student_id is unique for each student. So, we can use **student_id as a** key to identify each student without confusion.

- A **key** is used to **identify** rows in a table.
- It can be **one column** (like student_id) or **a group of columns**.
- It helps make the data **clear, accurate, and connected**.

## Primary Key

What it does:
- **A** primary key **uniquely identifies each row in a table.**
- **It** cannot have duplicate **values and** cannot be NULL**.**

**Rules:**

- Each table can have **only one** primary key.
- The primary key value must be **unique** and **not empty**.

**Example:**
Table: students

| student_id | Name | Age |
|---|---|---|
| 101 | Alice | 20 |
| 102 | Bob | 21 |
| 103 | Charlie | 22 |

Here, student_id is the **primary key** because it is unique for every student and it cannot be null.

```
CREATE TABLE students (
    student_id INTEGER PRIMARY KEY,
    name TEXT(100),
    age INTEGER
);
```

**StudentID** uniquely identifies each student.

**Characteristics:**

- Must be **unique**
- Cannot be **NULL**
- Only **one** primary key allowed per table

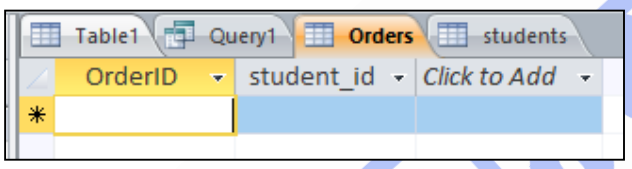
## Foreign Key

**Definition:**

A **Foreign Key** is a column in one table that **refers to the Primary Key** of another table. It creates a **relationship** between two tables.

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    student_id INT,
    FOREIGN KEY (student_id) REFERENCES students(student_id)
);
```

student_id in Orders refers to students.student_id.




**Characteristics:**

- Can have **duplicate values**
- Can be **NULL** (if allowed)
- Ensures **referential integrity**

## Candidate Key

**Definition:**

A **Candidate Key** is a column (or set of columns) that can **uniquely identify a record** in a table. Every table can have **multiple candidate keys**, but only one is chosen as the **Primary Key**.

**Example:**

In a User table:

UserID (unique)

Email (unique)
PhoneNumber (unique)

## Table Definition in SQL:

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    UserName VARCHAR(100),
    Email VARCHAR(100) UNIQUE,
    PhoneNumber VARCHAR(15) UNIQUE
);
```

**Explanation:**

- UserID, Email, and PhoneNumber are all **unique**, so they are **candidate keys**.
- **Primary Key** is chosen to be UserID.
- Email and PhoneNumber remain **alternate candidate keys** (not used as PK).

| UserID | UserName | Email | PhoneNumber |
|---|---|---|---|
| 1 | Alice | alice@example.com | 9876543210 |
| 2 | Bob | bob@example.com | 9123456789 |
| 3 | Charlie | charlie@example.com | 9988776655 |

All three can be candidate keys. You might choose UserID as the **primary key**, but Email and PhoneNumber are still **candidate keys**.

**Notes:**

- If later you decide to make Email the **Primary Key**, then UserID would become an **alternate key**.
- Candidate keys help you ensure **data consistency** by preventing duplicate entries for unique fields.

## Composite Key

**Definition:**

A **Composite Key** is a combination of two or more columns used together to uniquely identify a record.

## Concepts of SQL

**Characteristics:**

- Used when no single column is unique
- Each column **individually may not be unique**

**Example**

```
CREATE TABLE Enrollments (
    StudentID INT,
    CourseID INT,
    PRIMARY KEY (StudentID, CourseID)
);
```

Here, StudentID and CourseID together form a **composite key**.

Example Scenario:

### Student Enrollments in Courses

We want to track which **student** is enrolled in which **course**.

- A **student can enroll in multiple courses**
- A **course can have many students**
- But the **same student cannot enroll in the same course more than once**

Thus, we need to **uniquely identify each enrollment** using a combination of:

- StudentID
- CourseID

### SQL Table Definition with Composite Key:

```
CREATE TABLE Enrollments (
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    PRIMARY KEY (StudentID, CourseID) -- Composite Key
);
```

- Neither StudentID nor CourseID alone is unique.
- But the **combination** of both is **unique**, so it's a **Composite Key**.

| StudentID | CourseID | EnrollmentDate |
|---|---|---|
| 101 | 501 | 2025-06-01 |
| 101 | 502 | 2025-06-03 |
| 102 | 501 | 2025-06-04 |

**Explanation:**

- Student 101 is enrolled in two different courses — valid.
- Student 102 is also enrolled in Course 501 — valid.
- If you try to add (101, 501) again, it will violate the **composite primary key constraint**.
- **Composite Key** = StudentID + CourseID
- Used when **no single column is unique**, but the combination is.

## Alternate Key

**Definition:**

An **Alternate Key** is any **candidate key** that is **not chosen** as the primary key.

**Example:**

If both Email and UserID are unique in a Users table, and UserID is chosen as the primary key, then Email is an **alternate key**.

## SQL Table Definition:

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    UserName VARCHAR(100),
    Email VARCHAR(100) UNIQUE,
    PhoneNumber VARCHAR(15)
);
```

**Explanation:**

- UserID is the **Primary Key** - it uniquely identifies each user.
- Email is also **unique**, so it's a **Candidate Key**.
- Since we didn't choose Email as the primary key, it becomes an **Alternate Key**.

| UserID | UserName | Email | PhoneNumber |
|---|---|---|---|
| 1 | Alice | alice@example.com | 9876543210 |
| 2 | Bob | bob@example.com | 9123456789 |
| 3 | Charlie | charlie@example.com | 9988776655 |

## Super Key

**Definition:**

A **Super Key** is any combination of columns that **uniquely identifies a record.** Every candidate key is a super key, but not every super key is a candidate key (because it might have extra columns).

**Example:**

In a Students table:

- {StudentID} – Candidate Key and Super Key
- {StudentID, Name} – Super Key but **not a candidate key** (extra column)

## SQL Table Definition:

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100) UNIQUE,
    PhoneNumber VARCHAR(15) UNIQUE
);
```

| StudentID | Name | Email | PhoneNumber |
|---|---|---|---|
| 1 | Alice | alice@example.com | 9876543210 |
| 2 | Bob | bob@example.com | 9123456789 |
| 3 | Charlie | charlie@example.com | 9988776655 |

| Key Type | Description |
|---|---|
| Candidate Key | Minimal unique identifier (e.g., StudentID, Email) |
| Super Key | Any combination that uniquely identifies a record |
| Not Candidate | If it includes **extra** columns beyond what's necessary |

# Concepts of SQL

## Summary

| Key Type | Uniqueness | Allows NULLs | Can Have Multiple Per Table | Used For Relationships |
|---|---|---|---|---|
| Primary Key | Yes | No | No | Yes |
| Foreign Key | No | Yes | Yes | Yes |
| Candidate Key | Yes | No | Yes | No |
| Composite Key | Yes | No | Yes (composed of multiple) | Yes |
| Alternate Key | Yes | No | Yes | No |
| Super Key | Yes | Maybe | Yes | No |