ANKKIT KUMAR GUPPTA
DATA ANALYST | PROMPT ENGINEER

WITHOUT DATA, YOU'RE JUST ANOTHER PERSON WITH AN OPINION

Unlock the Secrets of Database Keys

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.

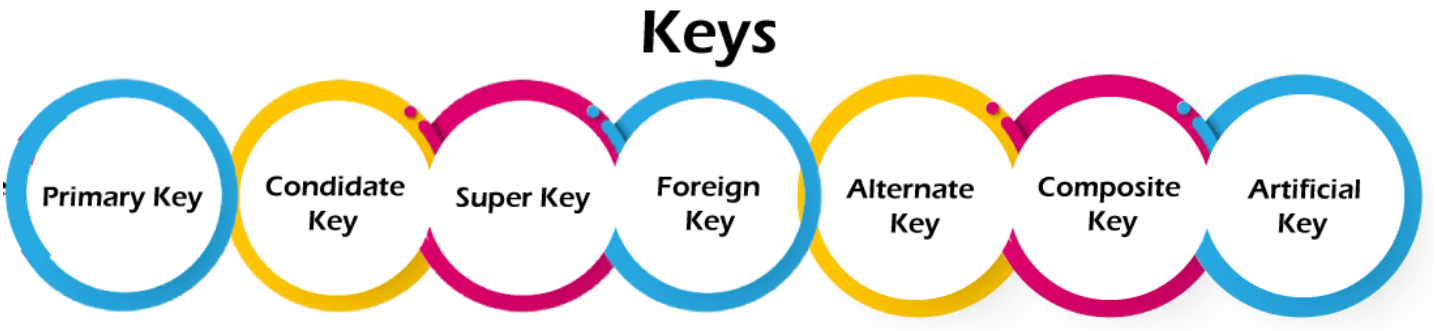
For example,

- 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.

STUDENT
ID
Name
Address
Course

PERSON
Name
DOB
Passport, Number
License_Number
SSN

Types of keys:



Primary key:

A **primary key** is a special column (or set of columns) in a table used to uniquely identify each record (row). It ensures there are no duplicate or missing entries for the data it represents.

Key Points About Primary Key:

1. **Uniqueness**: The primary key must be unique for every row in the table. No two rows can have the same value for the primary key.

2. **No Nulls**: A primary key cannot have a blank or null value. Every row must have a value for the primary key.

3. **Single Choice**: A table can have multiple columns that qualify to be unique (candidate keys), but only **one** is chosen as the primary key.

ID	Name	License_Number	Passport_Number
EM101	Alice	A12345	P98765
EM102	Bob	B67890	P87654
EM103	Charlie	C11223	P76543

- **ID** is a good choice for the primary key because it is unique for each employee.
- **License_Number** or **Passport_Number** could also work as primary keys because they are unique too.

However, **only one** of these columns is selected as the **primary key** based on the table's design and developer's requirements.

Why Is the Primary Key Important?

- It ensures that every record can be uniquely identified.
- It helps maintain data integrity and avoid duplication.

For example, if we use **ID** as the primary key, we can confidently reference employees using just their ID.

Query:

```
CREATE TABLE Employee ( ID VARCHAR(10) PRIMARY KEY, Name VARCHAR(50) NOT NULL, License_Number VARCHAR(20) UNIQUE, Passport_Number VARCHAR(20) UNIQUE );
```

Candidate key:

A **candidate key** is any column or combination of columns in a table that can uniquely identify each row (or record). These keys are potential candidates to be chosen as the **primary key** because they can uniquely identify rows, just like the primary key.

Once a primary key is chosen, the remaining candidate keys stay as **alternate keys**, but they are still unique and important.

Example: Consider an **EMPLOYEE** table.

ID	Name	SSN	Passport_Number	License_Number
EM101	Alice	123-45-6789	P98765	L12345
EM102	Bob	987-65-4321	P87654	L67890
EM103	Charlie	456-78-9012	P76543	L11223

In this table:

1. **ID**, **SSN**, **Passport_Number**, and **License_Number** can all uniquely identify rows.
2. Each of these attributes is a **candidate key** because they can be used to identify each employee.

If **ID** is chosen as the primary key, the remaining (SSN, Passport_Number, License_Number) will still remain **candidate keys**, and one of them could be used if needed.

Super Key:

A **super key** is any combination of columns (attributes) in a table that can uniquely identify a row (or record/tuple). It might contain more columns than necessary.

- A **candidate key** is a minimal version of a super key (the smallest set of columns needed to uniquely identify a row).
- All candidate keys are super keys, but not all super keys are candidate keys.

Example: Consider the **EMPLOYEE** table.

EMPLOYEE_ID	EMPLOYEE_NAME	SSN	Passport_Number
EM101	Alice	123-45-6789	P98765
EM102	Bob	987-65-4321	P87654
EM103	Alice	456-78-9012	P76543

1. **EMPLOYEE_ID** alone can uniquely identify a row. So, it's a **super key**.
2. A combination of **EMPLOYEE_ID** and **EMPLOYEE_NAME** (even though it's not necessary) can also uniquely identify a row. This is still a **super key**, but not a minimal one.
3. Other combinations like **EMPLOYEE_ID + SSN** or **EMPLOYEE_ID + Passport_Number** are also super keys.

In short:

- **Super key** = Any set of columns that uniquely identifies rows (can have extra columns).
- **Candidate key** = A minimal version of a super key (no extra columns).

Foreign key:

A **foreign key** is a column (or a group of columns) in one table that refers to the **primary key** in another table. It is used to link two tables together.

Key Points:

1. It helps establish a relationship between two tables.
2. The foreign key in one table matches the primary key in another table.
3. It ensures data consistency by allowing only valid values that exist in the referenced table's primary key.

Example:

1. **DEPARTMENT Table**: This table stores information about departments.

Department_Id	Department_Name
101	HR
102	IT
103	Finance

2. **EMPLOYEE Table**: This table stores information about employees.

Employee_Id	Employee_Name	Department_Id
EM101	Alice	101
EM102	Bob	102
EM103	Charlie	103

Here:

- **Department_Id** in the **DEPARTMENT table** is the **primary key**.
- **Department_Id** in the **EMPLOYEE table** is the **foreign key**, linking employees to their departments.

Why Use Foreign Keys?

If you try to assign a **Department_Id** in the **EMPLOYEE table** that doesn't exist in the **DEPARTMENT table**, the database will reject it. This ensures that the data stays consistent across both tables.

```
CREATE TABLE EMPLOYEE ( Employee_Id VARCHAR(10) PRIMARY KEY, Employee_Name VARCHAR(50) NOT NULL, Department_Id INT, FOREIGN KEY (Department_Id) REFERENCES DEPARTMENT(Department_Id) );
```

Alternate key:

An **alternate key** is a candidate key that **is not chosen** as the primary key.

How It Works:

1. **Candidate Keys**: These are the columns (or combinations of columns) that can uniquely identify each row in a table. A table can have more than one candidate key.

2. **Primary Key**: From the candidate keys, one is selected as the primary key.

3. **Alternate Key**: The remaining candidate keys, if any, are called **alternate keys**.

If there is only **one candidate key**, there are no alternate keys because all candidate keys must be either primary or alternate.

Example: Imagine an **Employee table** with the following candidate keys:

1. **Employee_Id** (a unique ID for each employee)
2. **PAN_No** (a unique number for each employee)

If you choose **Employee_Id** as the primary key, the other key, **PAN_No**, becomes the **alternate key**.

Employee_Id	PAN_No	Name	Department
101	ABCD1234P	John Doe	HR
102	WXYZ5678Q	Jane Smith	Finance
103	LMNO9101R	Bob Brown	IT

- **Primary Key**: Employee_Id
- **Alternate Key**: PAN_No

Key Points:

- **Candidate keys** can uniquely identify rows.
- One becomes the **primary key**; others (if any) are **alternate keys**.
- There are no alternate keys if there's only one candidate key.

Composite key:

A **composite key** is a primary key made up of **two or more columns** (attributes) instead of just one. It's used when a single column cannot uniquely identify a record in a table.

Why Do We Need Composite Keys?

- Sometimes, no single column is enough to make each row unique.
- In such cases, we combine two or more columns to form a **composite key**, which ensures every row is unique

Example: Imagine an **Employee table** where:

- An employee can have multiple roles (e.g., Developer, Tester).
- An employee can work on multiple projects.

This means you need to consider all three attributes—**Emp_ID**, **Emp_role**, and **Proj_ID**—together to uniquely identify each record.

Emp_ID	Emp_role	Proj_ID	Hours Worked
101	Developer	P001	20
101	Tester	P002	15
102	Manager	P001	25

Here:

- **Emp_ID** alone is not unique because the same employee has multiple roles.
- **Proj_ID** alone is not unique because multiple employees can work on the same project.
- **Emp_role** alone is not unique because the same role can exist for different employees.

So, the combination of **Emp_ID + Emp_role + Proj_ID** uniquely identifies each row. This is the **composite key**.

```
CREATE TABLE Orders ( OrderID INT, ProductID INT, Quantity INT, PRIMARY KEY (Emp_ID, Emp_role, Proj_ID) );
```

Artificial key:

An **artificial key** is a special kind of key that is **created manually** when:

1. The natural primary key (like a combination of multiple columns) is **too large or complex**.
2. There is no simple, unique identifier for each row in the table.

Instead of relying on existing data, an artificial key assigns a **new, unique value** (usually a number in serial order) to each row. This makes it easier to manage and relate records across tables.

Why Use an Artificial Key?

- Simplifies data management.
- Avoids using long or complicated primary keys.
- Makes relationships between tables more straightforward.

Example:

Let's say you have an **Employee table** where the natural primary key would be a combination of **Emp_ID**, **Emp_role**, and **Proj_ID** (because this combination is unique for each row).
This combination is cumbersome, so you create a new column, **Emp_Key**, with unique values like **1, 2, 3, etc.**, to act as the primary key instead.

Emp_Key	Emp_ID	Emp_role	Proj_ID
1	101	Developer	P001
2	102	Tester	P002
3	101	Team Lead	P003

Here, **Emp_Key** is the **artificial key** created to uniquely identify rows and simplify the table structure.

****

Surrogate Key:

A **surrogate key** is an extra column added to a database table that contains unique, system-generated values. These values have no business meaning—they're simply used to uniquely identify each row.

Characteristics of Surrogate Keys:

1. **Generated by the System:** The database generates the value (e.g., auto-increment numbers, GUIDs).
2. **No Business Meaning:** The value is unrelated to the actual data in the table.
3. **Uniqueness:** Ensures each row in the table is uniquely identifiable.
4. **Consistency:** Does not change even if business rules or data values change.

Example of Surrogate Key: Imagine a **Student Table** without a natural key.

Student Name	Grade
John Doe	A
Jane Smith	B
John Doe	A

Here, there's no unique identifier for rows (two students have the same data). A **surrogate key** can solve this:

Student ID	Student Name	Grade
1	John Doe	A
2	Jane Smith	B
3	John Doe	A

- **Student ID** is the surrogate key. It’s an auto-incrementing integer, ensuring each row is unique.

Natural Key:

A **natural key** is a column or combination of columns in a table that uniquely identifies each row **based on the actual data**. This key has **business meaning** and reflects real-world identifiers.

Characteristics of Natural Keys:

1. **Derived from the Data:** Comes from existing table attributes.
2. **Business Meaning:** The value has relevance to the data.
3. **May Change Over Time:** If business rules change, the key might need updates.

Example of Natural Key: Consider a **Student Table** with **Registration Number** as a unique column.

Registration No	Student Name	Grade
210101	John Doe	A
210102	Jane Smith	B
210103	John Doe	A

- **Registration No** is the natural key because it uniquely identifies each student and has business meaning.

Comparison of Surrogate & Natural Keys:

Aspect	Surrogate Key	Natural Key
Definition	System-generated unique identifier.	Derived from actual table attributes.
Business Meaning	No business meaning.	Has business meaning.
Changes Over Time	Rarely changes, if ever.	May change due to business requirements.
Example	Auto-increment ID or GUID.	SSN, Registration Number, or Email.

When to Use Each Key?

- **Use Surrogate Keys:**
  - When no natural unique attribute exists.
  - When performance is a priority (smaller surrogate keys lead to faster queries).
  - To avoid dealing with changes in business rules.

- **Use Natural Keys:**
  - When there's a meaningful, unique column (e.g., SSN, ISBN for books).
  - To reduce the need for additional columns.
  - When joins and lookups benefit from meaningful data.

ANKKIT KUMAR GUPPTA

DATA ANALYST | PROMPT ENGINEER

WITHOUT DATA, YOU'RE JUST ANOTHER PERSON WITH AN OPINION

If you would like to learn more & stay updated, please follow me on LinkedIn