Key:

In a relational database, **keys** are fundamental concepts that define how records within a table or across different tables are uniquely identified, related, and maintained.

Importance of Keys in Relational Databases:

- **Data Integrity:** Keys help maintain the accuracy and consistency of data in a relational database.
- **Relationship Management:** Foreign keys allow the establishment of relationships between tables, enabling complex queries and data retrieval.
- **Efficient Data Retrieval:** Indexes are often created on key columns, which improves the speed of data retrieval operations.
- **Avoiding Redundancy:** Proper use of keys helps in normalization, reducing data redundancy and ensuring that the database is logically structured.

1. Primary Key

Definition: A primary key is a column, or a set of columns, that uniquely identifies each row in a table. A table can have only one primary key, and the values in the primary key column(s) must be unique and cannot be `NULL`.

Example: Consider a table `students`:

StudentID	FirstName	LastName
101	John	Doe
102	Jane	Smith
103	Bob	Brown

Here, `StudentID` is the primary key because it uniquely identifies each student.

2. Foreign Key

Definition: A foreign key is a column, or a set of columns, in one table that refers to the primary key in another table. The foreign key creates a relationship between the two tables, enforcing referential integrity.

Example: Consider two tables, `students` and `enrollments`:

Students Table:

StudentID	FirstName	LastName
101	John	Doe
102	Jane	Smith

Enrollments Table:

EnrollmentID	StudentID	CourseID
1	101	C101
2	102	C102

In the `enrollments` table, `StudentID` is a foreign key that references the `StudentID` in the `students` table.

3. Candidate Key

Definition: A candidate key is a column, or a set of columns, that can uniquely identify each row in a table. A table can have multiple candidate keys, but one of them will be chosen as the primary key.

Example: In a table `employees`, both `EmployeeID` and `SocialSecurityNumber` (SSN) could be candidate keys, as both can uniquely identify each employee.

EmployeeID	SSN	FirstName	LastName
1	123-45-6789	Alice	Johnson
2	987-65-4321	Bob	Smith

Here, both `EmployeeID` and `SSN` are candidate keys, but typically, `EmployeeID` might be chosen as the primary key.

4. Alternate Key

Definition: An alternate key is any candidate key that is not chosen as the primary key. It can be used to identify a row uniquely but is not the primary means of doing so.

Example: Continuing from the previous example, if `EmployeeID` is chosen as the primary key, then `SSN` becomes an alternate key.

5. Composite Key

Definition: A composite key is a primary key that consists of two or more columns used together to uniquely identify a row in a table.

Example: In a table `order_items` that stores items in an order:

OrderID	ProductID	Quantity
1001	P001	2
1001	P002	1
1002	P001	4

Here, the combination of `OrderID` and `ProductID` forms a composite key, ensuring that each product in an order is uniquely identified.

6. Super Key

Definition: A super key is any combination of columns that can uniquely identify a row in a table. It is a broader concept than a candidate key, as it can include additional columns that are not necessary for uniqueness.

Example: In the `students` table, `StudentID`, `FirstName`, and `LastName` together form a super key, even though `StudentID` alone is sufficient to uniquely identify a row.

7. Unique Key

Definition: A unique key is similar to a primary key in that it enforces the uniqueness of the column(s) it is applied to, but unlike a primary key, a table can have multiple unique keys, and a unique key can accept `NULL` values.

Example: In an `employees` table:

EmployeeID	Email
1	john@example.com
2	jane@example.com

Here, `Email` could be a unique key because each employee must have a unique email address.

Functional Dependency:

Definition: A functional dependency occurs when one attribute uniquely determines another attribute. If you know the value of one attribute, you can uniquely determine the value of another.

Notation: If attribute B is functionally dependent on attribute A, it is denoted as A→B. This means that for every unique value of A, there is only one corresponding value of B.

Example: Consider a table storing employee data:

EmployeeID	EmployeeName	Department
101	John Doe	Sales
102	Jane Smith	HR
103	Bob Brown	IT

In this table, "EmployeeID" functionally determines "EmployeeName" and "Department" because each "EmployeeID" corresponds to a unique "EmployeeName" and "Department."

Partial Dependency

Definition: A partial dependency occurs when a non-key attribute is functionally dependent on part of a composite primary key (but not on the whole key).

Example: Consider a table storing order details with a composite primary key made up of "OrderID" and "ProductID":

OrderID	ProductID	ProductName	Quantity
1001	P001	Widget	10
1001	P002	Gadget	5
1002	P001	Widget	7

Here, "ProductName" is partially dependent on "ProductID" but not on the entire composite key ("OrderID" and "ProductID"). This violates the rules of 2NF, as all non-key attributes should depend on the entire key.

Transitive Dependency

Definition: A transitive dependency occurs when one non-key attribute is dependent on another non-key attribute, which is itself dependent on the primary key.

Example: Consider the following table:

EmployeeID	Department	DepartmentHead
101	Sales	Mr. White
102	HR	Mrs. Black
103	IT	Mr. Green

In this table, "DepartmentHead" is dependent on "Department," which is dependent on "EmployeeID." This creates a transitive dependency, which violates the rules of 3NF.

Normalization:

Database Normalization is a process used in database design to organize data to reduce redundancy and improve data integrity.

It involves dividing a database into tables and defining relationships between them according to specific rules.

These rules are applied through various "normal forms," which are the stages of normalization. The most commonly used normal forms are the First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF).

1st Normal Form (1NF): Eliminate Repeating Groups

Rule: A table is in 1NF if it contains only atomic (indivisible) values and each column contains values of a single type.

Example: Consider a table that stores information about students and the courses they are enrolled in

StudentID	StudentName	Courses
101	John Doe	Math, Science, History
102	Jane Smith	Math, English
103	Sam Brown	History, Art

Issue: The `Courses` column contains multiple values, which violates the rule of 1NF that requires each column to contain atomic (indivisible) values.

1NF Solution: To bring this table into 1NF, we create a new row for each course a student is enrolled in, ensuring that each column contains only atomic values.

StudentID	StudentName	Course
101	John Doe	Math
101	John Doe	Science
101	John Doe	History
102	Jane Smith	Math
102	Jane Smith	English
103	Sam Brown	History
103	Sam Brown	Art

2nd Normal Form (2NF): Eliminate Partial Dependencies

Rule: A table is in 2NF if it is in 1NF and all non-key attributes are fully dependent on the primary key.

Example: Now that the table is in 1NF, suppose we want to store additional information about the courses, such as the instructor. The table might look like this:

StudentID	StudentName	Course	Instructor
101	John Doe	Math	Mr. Smith
101	John Doe	Science	Dr. Johnson
101	John Doe	History	Ms. Brown
102	Jane Smith	Math	Mr. Smith
102	Jane Smith	English	Ms. Davis
103	Sam Brown	History	Ms. Brown
103	Sam Brown	Art	Mr. White

Issue: The `Instructor` column depends on the `Course` rather than the combination of `StudentID` and `Course`. This violates the 2NF rule, which requires that all non-key attributes be fully dependent on the primary key.

2NF Solution: We decompose the table into two smaller tables: one for students and their courses, and another for courses and their instructors.

Students_Courses Table:

StudentID	StudentName	Course
101	John Doe	Math
101	John Doe	Science
101	John Doe	History
102	Jane Smith	Math
102	Jane Smith	English
103	Sam Brown	History
103	Sam Brown	Art

Courses_Instructors Table:

Course	Instructor
Math	Mr. Smith
Science	Dr. Johnson
History	Ms. Brown
English	Ms. Davis
Art	Mr. White

3rd Normal Form (3NF): Eliminate Transitive Dependencies

Rule: A table is in 3NF if it is in 2NF and all the attributes are only dependent on the primary key, meaning no transitive dependencies exist.

Example: Let's assume we add another attribute, `Department`, to the `Courses_Instructors` table, indicating the department responsible for each course:

Course	Instructor	Department
Math	Mr. Smith	Science
Science	Dr. Johnson	Science
History	Ms. Brown	Humanities
English	Ms. Davis	Humanities
Art	Mr. White	Arts

Issue: The `Department` column depends on the `Course`, which depends on the `Instructor`, creating a transitive dependency. This violates the 3NF rule, which requires that all attributes depend only on the primary key.

3NF Solution: We separate the department information into its own table and remove the transitive dependency.

Courses Table:

Course	Department
Math	Science
Science	Science
History	Humanities
English	Humanities
Art	Arts

Instructors Table:

Instructor	Course
Mr. Smith	Math
Dr. Johnson	Science
Ms. Brown	History
Ms. Davis	English
Mr. White	Art

Now, each table is in 3NF, as all columns are functionally dependent on the primary key, and there are no transitive dependencies.

Summary of Normalization Steps

- **1NF:** Ensure that each column contains atomic values and that there are no repeating groups.
- **2NF:** Ensure that all non-key attributes are fully dependent on the entire primary key (eliminate partial dependencies).
- **3NF:** Ensure that there are no transitive dependencies; all attributes should depend only on the primary key.

Benefits of Normalization:

- **Eliminates redundancy:** Reduces duplicate data, saving storage space and reducing the chance of inconsistency.
- **Improves data integrity:** Ensures that data modifications (insertions, updates, deletions) occur without anomalies.
- **Enhances query performance:** Simplifies complex queries and makes them more efficient.
- **Simplifies maintenance:** Easier to update, delete, or insert data without affecting the rest of the database.