In SQL, keys and constraints are fundamental components used to ensure data integrity and establish relationships between tables. This article will explain the primary types of keys and constraints with practical examples.

Keys in SQL

Keys are crucial for identifying records within a table and establishing relationships between different tables. Here are the key types we'll cover:

Primary Key

A primary key is a unique identifier for each record in a table. It cannot contain NULL values and must contain unique values.

```
DROP TABLE IF EXISTS Employees;
CREATE TABLE IF NOT EXISTS Employees (
  employee_id SERIAL PRIMARY KEY,
  name VARCHAR(50),
  department VARCHAR(50)
);
SELECT * FROM Employees;
```

In this example, the employee_id column is the primary key for the Employees table. The SERIAL type automatically generates a unique value for each new record.

Composite Key

A composite key consists of two or more columns that together uniquely identify a record.

```
DROP TABLE IF EXISTS Orders;

CREATE TABLE Orders (

  order_id SERIAL,

  customer_id INT,

  order_date DATE,

  PRIMARY KEY (order_id, customer_id)

);

SELECT * FROM Orders;
```

Here, order_id and customer_id together form the composite primary key for the Orders table, ensuring each order is uniquely identified by both columns.

Constraints in SQL

Constraints enforce rules on data columns to maintain data integrity. Here are the common types of constraints:

Foreign Key

A foreign key is a column (or a set of columns) that establishes a link between the data in two tables. It ensures referential integrity by making sure that the value in the foreign key column exists in the referenced primary key column.

```
DROP TABLE IF EXISTS Orders;

CREATE TABLE Orders (
  order_id SERIAL PRIMARY KEY,
  customer_id INT,
  order_date DATE,
  FOREIGN KEY (customer_id) REFERENCES Employees (employee_id)
);

SELECT * FROM Orders;
```

In this example, customer_id in the Orders table is a foreign key that references the employee_id column in the Employees table.

Unique Constraint

A unique constraint ensures that all values in a column are distinct.

```
DROP TABLE IF EXISTS Employees;

CREATE TABLE IF NOT EXISTS Employees (
  employee_id SERIAL PRIMARY KEY,
  email VARCHAR(50) UNIQUE,
  name VARCHAR(50),
  department VARCHAR(50)
);

SELECT * FROM Employees;
```

Here, the email column in the Employees table must contain unique values, preventing duplicate email entries.

Composite Unique Constraint

A composite unique constraint ensures that the combination of values in two or more columns is unique across all records.

```
DROP TABLE IF EXISTS Employees;

CREATE TABLE IF NOT EXISTS Employees (
  employee_id SERIAL PRIMARY KEY,
  email VARCHAR(50) UNIQUE,
  phone INT UNIQUE,
  name VARCHAR(50),
  department VARCHAR(50),
  UNIQUE (email, phone)
);

SELECT * FROM Employees;
```

This example ensures that the combination of email and phone values is unique in the Employees table.

Check Constraint

A check constraint enforces a condition on the values in a column. If a record violates the condition, it is not allowed to be added to the table.

```
DROP TABLE IF EXISTS Employees;

CREATE TABLE IF NOT EXISTS Employees (
  employee_id SERIAL PRIMARY KEY,
  email VARCHAR(50) UNIQUE,
  age INT CHECK (age >= 18),
  phone INT UNIQUE,
  name VARCHAR(50),
  department VARCHAR(50),
  UNIQUE (email, phone)
);

SELECT * FROM Employees;
```

In this table, the age column must contain values greater than or equal to 18. This ensures that no employee can be younger than 18 years old.

Conclusion

Understanding keys and constraints is essential for designing robust and efficient databases. Keys ensure each record can be uniquely identified, while constraints enforce rules to maintain data integrity. By using these tools effectively, you can create databases that are both reliable and scalable.