# Designing a Database

## Database-Schema

a *Database Schema* describes the structure of a database. Database schemas generally contain information about table/column names, data types/constraints, relationships between tables, and user roles.

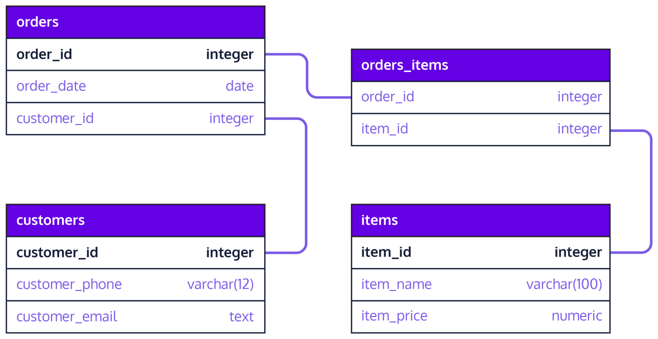
## PostgreSQL Naming Conventions

There are several naming conventions in PostgreSQL:

- Column names should be lower case with underscores between words (eg., `birth_date` ).
- The primary key of a table is often named `id` .

## Database Schema Design Tools

The image above shows an example database schema that was illustrated using a design tool. The schema contains four tables named orders, customers, orders_items, and items, respectively. The lines in the visualization show how columns in different tables are related to each other.



## Database Tables

Database schemas describe the number of tables in a database and the data that they contain. Database tables should usually relate to a single construct described by an identifier called a primary key. For example, in an `orders` table where every order has a customer, it would be inefficient to write the complete details of each customer for every order they made. Instead, a separate `customers` table could contain that information.

## PostgreSQL Variable Types

When designing a database schema in PostgreSQL, every column must have a data type. For example, the table created in the example code has three columns with types `integer` , `varchar` , and `boolean` , respectively. This helps preserve data integrity over time by restricting the data that can be entered into the table.

```
CREATE TABLE people (
    age integer,
    name varchar,
    is_citizen boolean
);
```

## Primary Keys

The primary key of a database table is a column or group of columns that can be used to uniquely identify every row of the table. For example, a table of students might have a primary key named `student_id` , which contains unique ID numbers for each student.

## Composite Primary Keys

A primary key made up of multiple columns is called a composite primary key. Composite primary keys can be used when no single column uniquely identifies a row of a table, but multiple columns do. For example, the example table shown here could have a composite primary key made up of `student_id` and `course_id` .

```
| student_id| course_id | grade |
|-----------|-----------|-------|
| 1         | 503       | A     |
| 1         | 401       | A-    |
| 2         | 503       | B     |
```

## Foreign Keys

Database tables can have foreign key(s), which reference the primary key of another table. For example, an `orders` table may contain a column named `customer_id` , which references an `id` column in a separate `customers` table. In this case, the `customer_id` column in the `orders` table could be designated as a foreign key.

## Creating a Primary Key

In PostgreSQL, a column can be designated as a primary key using the `PRIMARY KEY` keyword. For example, the example code shows two `CREATE TABLE` statements. In the `orders` table, `order_id` is designated as a primary key. In the `orders_items` table, `order_id` and `item_id` are used to designate a composite primary key.

```
CREATE TABLE orders(
order_id integer PRIMARY KEY,
order_date date,
);

CREATE TABLE orders_items(
order_id integer,
order_date date,
item_id integer,
PRIMARY KEY (order_id,item_id)
);
```

## Creating a Foreign Key

In PostgreSQL a foreign key is defined by using the `REFERENCES` keyword. For example, the code here shows two `CREATE TABLE` statements. The `orders` table contains a foreign key called `customer_id` , which references the `id` column of the `customers` table.

```
CREATE TABLE orders (
  order_number integer,
  customer_id integer REFERENCES
customers(id),
);

CREATE TABLE customers (
  id integer PRIMARY KEY,
  name text
);
```

## One-to-One Database Relationships

In a relational database, two tables have a one-to-one relationship if each row in one table links to exactly one row in the other table, and vice versa. For example, a table of `employees` and a table of `employee_contact_info` might have a one-to-one relationship if every employee listed in the `employees` table has contact information listed in the `employee_contact_info` table and vice versa.

## Many-to-One Database Relationships

In a relational database, two tables have a many-to-one relationship if each row in one table links to multiple rows of the other table. For example, a table of `customers` and a table of `orders` would have a many-to-one relationship if each customer can make multiple orders, but each order can only be associated with one customer.

## Many-to-Many Database Relationships

In a relational database, two tables have a many-to-many relationship if each row in one table can link to multiple rows in the other table, and vice versa. For example, a table of `songs` and a table of `artists` would likely have a many-to-many relationship because songs can have multiple artists and artists can have multiple songs.

## Implementing a Many-to-One Database Relationship

A many-to-one relationship can be implemented in PostgreSQL by creating a foreign key that references the primary key of another table. For example, the code here implements a many-to-one relationship between an `orders` and a `customers` table, where each customer can be associated with multiple orders.

```
CREATE TABLE orders (
  order_number integer,
  customer_id integer REFERENCES
customers(id),
);

CREATE TABLE customers (
  id integer PRIMARY KEY,
  name text
);
```

## Implementing a Many-to-Many Database Relationship

A many-to-many database relationship can be implemented in PostgreSQL using a third cross-reference table. This table should have two foreign keys referencing the primary keys of the related tables, as well as a composite primary key made up of the foreign key columns. For example, the code here implements a many-to-many relationship between a `songs` and an `artists` table.

```
CREATE TABLE songs (
  id integer PRIMARY KEY,
  name varchar(100)
);

CREATE TABLE artists (
  id integer PRIMARY KEY,
  name varchar(100)
);

CREATE TABLE songs_artists (
  artist_id integer REFERENCES
artists(id),
  song_id integer REFERENCES songs(id),
  PRIMARY KEY (artist_id, song_id)
);
```

## Implementing a One-to-One Database Relationship

A one-to-one relationship can be enforced in PostgreSQL by first creating a many-to-one relationship via a foreign key, then implementing a `UNIQUE` constraint on the foreign key. For example, the code here implements a one-to-one relationship between tables named `employees` and `contact_info`.

```
CREATE TABLE employees (
  id integer PRIMARY KEY,
  name varchar(100)
);

CREATE TABLE contact_info (
  employee_id integer REFERENCES
employees(id) UNIQUE,
  email text,
  phone_number varchar(9)
);
```