codecademy

SQL Concepts →

Docs / SQL / **Primary Keys**

Primary Keys


+6

Published Aug 4, 2021 • **Updated Mar 20, 2023**

Contribute to Docs →

Primary keys are special columns that are used to uniquely identify each row of a table in SQL.

Syntax

```
CREATE TABLE table_key (
  id INTEGER PRIMARY KEY,
  column_1 TEXT,
  column_2 INTEGER
);
```

The `PRIMARY KEY` constraint is used to create columns that uniquely identify each row. A primary key column has a few requirements:

- None of the values can be `NULL`.
- Each value must be unique (e.g., two rows in a `customers` table wouldn't have the same primary `customer_id`).
- A table cannot have more than one primary key.

Attempts to insert a row with an existing primary key will result in a constraint violation that prevents the new row from being added.

If a table was created without a primary key, it can be added with the `ALTER TABLE` command. The statement below adds a primary `id` column, via the `PRIMARY KEY` constraint, to `table_name`:

```
ALTER TABLE table_name
ADD PRIMARY KEY (id);
```

Foreign Keys

When the primary key for one table appears in a different table, it is called a foreign key. The most common types of joins will be joining a foreign key from one table with the primary key from another table.

Using the following `customers` table as an example:

```
CREATE TABLE customers (
  customer_id INTEGER NOT NULL,
  first_name varchar(255),
  last_name varchar(255)
);
```

The `orders` table is created and joined via `FOREIGN KEY` with the existing `customer` table through its `customer_id`:

```
CREATE TABLE orders (
  order_id INTEGER NOT NULL,
  total_cost FLOAT,
  purchase_date DATE,
  customer_id INTEGER NOT NULL,
  PRIMARY KEY (order_id),
  FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

The displayed `orders` table, with its primary key (`order_id`) and foreign key

(`customer_id`), may look like this:

order_id	customer_id	total_cost	purchase_date
1	1001	13.99	2022-01-01
2	1294	61.42	2022-01-01
3	1001	23.45	2022-01-02

Composite Keys

Sometimes, having one primary key per table is not enough to uniquely identify a row. In such cases, multiple columns would work as composite keys for the table. This requirement should be detected during the designing phase of a database.

For example, a database of car parts will have to uniquely identify a row of parts. Either the `engine_ID` or `body_ID` could be used. However, this may create ambiguity as cars could get their engines swapped.

Depending on local regulations, a car may require an engine part ID and a body ID to be associated with a license plate. One solution might be adding more row information about the car, such as `left_door_ID` , `gearbox_ID` , etc. But then a specific car would have to be identified by two different aspects: its body and its engine.

A composite key would be useful in this case. This is how a `vehicle_registry` table might look (extra parts/columns omitted for brevity):

engine_id	body_id	gearbox_id	purchase_date
500	abc	001	2022-01-01
600	def	002	2022-01-01
700	ghi	003	2022-01-02

The statement below creates the `vehicle_registry` table with a composite key:

```
CREATE TABLE vehicle_registry (
  engine_id INTEGER,
  body_id TEXT,
  gearbox_id INTEGER,
  purchase_date DATE,
```

```
  PRIMARY KEY(engine_id, body_id, purchase_date)
);
```

All contributors

- @BrandonDusch
- @Fedoteh
- Anonymous contributor
- @KyraThompson
- @rclarkeweb
- @christian.dinh
- @Victoria-DR

Looking to contribute?

- Learn more about how to get involved.
- Edit this page on GitHub to fix an error or make an improvement.
- Submit feedback to let us know how we can improve Docs.

Learn SQL on Codecademy

Learn SQL

In this SQL course, you'll learn how to manage large datasets and analyze real data using the standard data management language.

Beginner Friendly **4** Lessons

Back to top