# Understanding Relational Databases

Simply put, you can think of databases as software that simply stores and organizes data in an efficient way. When we think of databases, we tend to think of what are called Relational Databases because they store data in a *relational* way. This means the data is related to each other in some way. For example, a relational database may have several tables that help a product ordering application. These tables may be named something like *orders, customers, addresses, payments,* etc... You would then be able to connect these tables through what are called primary and foreign keys.

## Primary Keys

Primary keys are unique for each record in a database. Lets use this website for example and assume that we have users with the following names:

| **NAME** |
| --- |
| John |
| Lauren |
| Carl |

We might store this data in a table called *users*. Well, just having one table in our websites database is not very useful. Say we also want to see what these users are posting we could also have a *posts* table that may look like the following:

| **POST** |
| --- |
| Understanding Redshift |
| Seattle |
| AWS |

Our database now has two *ENTIRE* tables! They are not very useful, though, because we cannot see who is posting what. This is where *primary keys* become useful. Now, we could setup just one large table that contains all of the websites data, but this quickly grows unwieldy whenever the data models become more complex than a simple blog. So instead we can assign a number that increases by 1 each time we add a new user or post. So we can refactor the tables to look like the following.

| **user_id** (This will be the name of our primary key for users) | **name** |
|---|---|
| 1 | John |
| 2 | Lauren |
| 3 | Carl |

| **post_id** (This will be the name of the primary key for posts) | **post_title** |
|---|---|
| 1 | Understanding Redshift |
| 2 | Seattle |
| 3 | AWS |

We now have our tables organized with *primary keys*, but we are unable to join the two tables to see who posted what article. For that, we will need to make use of a *foreign key*.

# Foreign/Secondary Keys

A *foreign key* is a column or a set of columns in a table whose values correspond to the values of the primary key in another table. In order to add a row with a given foreign key value, there must exist a row in the related table with the same primary key value.

For our example a good foreign key for the *posts* table would be *user_id* so that way we can easily see who posted what article. So we can refactor our table to look something like this:

| **post_id** (This will be the name of the primary key for posts) | **post_title** | **user_id** |
|---|---|---|
| 1 | Understanding Redshift | 1 |
| 2 | Seattle | 1 |
| 3 | AWS | 2 |

We are now able to easily see who posted what article by matching the *user_id* to the *users* table. Our SQL query would look something like this:

```
SELECT
    p.post
    , u.name
FROM posts p

JOIN users u
    -- Join the foreign key of user_id from post to the primary key of user_id in users
    ON p.user_id = u.user_id
```

Our results would then look like the following:

| post | name |
| --- | --- |
| Understanding Redshift | John |
| Seattle | John |
| AWS | Lauren |

Revision #8
Created 13 June 2022 22:01:53 by Trevor Barnes
Updated 14 June 2022 20:25:18 by Trevor Barnes