Interactive - 08 - create unique id and a primary key

It is really hard to update data when you don't have a unique way of identifying a row. People will often imagine that there is something unique in the data. Some combination of name/date/place-of-birth, generally it is not true.

The easy way to deal with this is to create unique keys. PostgreSQL offers two ways of doing this. The first is a key that counts up. This makes short keys, that's good but also creates a bottle neck with a single source of "count-up" values. The second way is use a university unique identifier or UUID. UUIDs are can be created in parallel and will not collide. This has lot's of advantages, but they are long, hard to type and take up lots of space in your database.

We will get back to sequential values later, for a different purpose. Let's do the more general approach and use UUIDs.

We are going to change our table to have a unique ID column, then create an index on that column.

This relies on having a way to generate UUIDs, it is an extension package to the default PostgreSQL database. (double quotes, " are correct for this command).

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
```

There are ways to add a column to the table without re-creating the table. They will not work for this.

Let's save our data for later, then recreate the table.

First we are going to rename the table, so we can save it for later. (Familiar, we did this last time)

```
drop table if exists old_name_list;
ALTER TABLE name_list
    RENAME TO old_name_list;
```

Now we will re-create the table with the check constraint.

```
create table name_list (
    name_list_id UUID NOT NULL DEFAULT uuid_generate_v4() primary key,
    real_name text,
    age int check ( age > 0 and age < 154 ),
    state char varying (2)
);
```

Now copy the data back.

```
INSERT INTO name_list ( real_name, age, state )
    SELECT real_name, age, state
    FROM old_name_list;
```

Verify data amount. Check how many rows.

```
select count(1) from name_list;
select count(1) from old_name_list;
```

Provided that the counts are the same let's clean up the temporary table with the old data.

```
drop table old_name_list;
```

There are 2 parts that we added. Both a ID column that is unique and a thing called a `PRIMARY KEY`.

Your result should be similar to this list (the name_list_id's will be unique to you and will not match the UUIDs below).

```
             name_list_id             |      real_name      | age | state
--------------------------------------+---------------------+-----+-------
 1fb069ca-a4a2-412a-8401-c835b6c45cb7 | Jane True           |  20 | WY
 39fff492-f369-4552-bf49-b8d7ae57bf9a | Tom Ace             |  31 | NJ
 8e91d65f-ee0a-4642-8d62-1cc1e1058238 | Steve Pen           |  33 | NJ
 e245b3ec-9e5f-4112-ac7e-1d9732d9a4f1 | Laura Jean Alkinoos |  34 | PA
 b8c2468e-694d-46de-acc3-e8da239aa1a3 | Philip Schlump      |  62 | WY
 983b17d2-39f7-45bc-96b2-66ffdbb96c0e | Bob True            |  22 | WY
 2db6af28-47c7-4107-9b67-bdec3321f14c | Liz Trubune         |  30 | WY
 ea51c6b2-37f0-47ed-9211-d1064d3eacfd | Jane True           |  44 | WY
 4c827537-1755-4242-8241-a77ff4173554 | Lary Smith          |  58 | NJ
 3674b39d-4a61-45e0-99e6-65fb7edbbf82 | Dave Dave           |  21 | NJ
 e543c622-24be-4cc2-8241-e8a2dfd1f15c | Laura Ann Alkinoos  |  34 | PA
(11 rows)
```

The grading code will look at the table name_list and verify that you have 11 rows of data in it.

Tags: "primary key","uuid","unique id","UUID"

Validate: SQL-Select,"select 'PASS' as x from (select count(1) as x from name_list) as t1 where t1.x = 11"