Chapter 6

October 29, 2019

```
In [1]: %reload_ext sql
```

```
In [2]: %sql postgresql://postgres:postgres@localhost:5432/analysis
```

```
Out[2]: 'Connected: postgres@analysis'
```

1 Joining Tables in a Relational Database

1.1 Linking Tables Using JOIN

- To connect tables in a query, we use *JOIN . . . ON* statement
  - this is one of JOIN variants
- the JOIN statement links one table to another in the database during a query
- SELECT
  - *
- FROM
  - table_a
- JOIN
  - table_b
- ON
  - table_a.key_column = table_b.foreign_key_column
- This is similar to the SELECT syntax
- we simply bring together two tables with the FROM and JOIN keywords
- the *ON* keyword is the specifier for the columns we want to use to match values
- when the query runs it examines both tables and then returns columns from both tables where the values match in the columns specified in the ON clause

```
In [5]: %%sql

        CREATE TABLE
            departments (
```

```
            dept_id bigserial,
            dept varchar(100),
            city varchar(100),
            CONSTRAINT dept_key PRIMARY KEY (dept_id),
            CONSTRAINT dept_city_unique UNIQUE (dept, city)
            );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

Out[5]: []

- we create a table
- we define the unique primary key as dept id
- the **UNIQUE** keyword guarantees, that the values or the combination of values in more than one column are unique
  - here each **pair** of dept and city has to be unique

In [6]:
```
%%sql

        CREATE TABLE
            employees (
            emp_id bigserial,
            first_name varchar(100),
            last_name varchar(100),
            salary integer,
            dept_id integer REFERENCES departments (dept_id),
            CONSTRAINT emp_key PRIMARY KEY (emp_id),
            CONSTRAINT emp_dept_unique UNIQUE (emp_id, dept_id)
            );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

Out[6]: []

- we create a table
- we define the unique primary key as emp id
- we also hava a reference to another id which is a **foreign key** (it is the key of the departments table, we define this with the **REFERENCES** keyword
  - a foreign key can be empty and contain duplicates
- note that the dept_id is a reference, a **foreign key** which references to the departments table
  - here each **pair** of empt_id and dept_id has to be unique

```
In [7]: %%sql

        INSERT INTO
            departments (dept, city)
        VALUES
            ('Tax', 'Atalanta'),
            ('IT', 'Boston');

 * postgresql://postgres:***@localhost:5432/analysis
2 rows affected.


Out[7]: []
```

```
In [8]: %%sql

        INSERT INTO
            employees (first_name, last_name, salary, dept_id)
        VALUES
            ('Nancy', 'Jones', 62500, 1),
            ('Lee', 'Smith', 59300, 1),
            ('Soo', 'Nguyen', 83000, 2),
            ('Janet', 'King', 95000, 2);

 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.


Out[8]: []
```

- note that the dept_id is a reference, a **foreign key** which references to the departments table

```
In [9]: %%sql

        SELECT * FROM departments;

 * postgresql://postgres:***@localhost:5432/analysis
2 rows affected.


Out[9]: [(1, 'Tax', 'Atalanta'), (2, 'IT', 'Boston')]
```

- the **dept_id** column is the tables primary key
- a primary key is a column or collection of columns whose values uniquely identify each row in a table
  - they must have a unique value for each row
  - the column or collection of columns can't have missing values
- you define the primary key with the **CONSTRAINT** keyword, which will be covered later

```
In [10]: %%sql

         SELECT * FROM employees;

 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.


Out[10]: [(1, 'Nancy', 'Jones', 62500, 1),
          (2, 'Lee', 'Smith', 59300, 1),
          (3, 'Soo', 'Nguyen', 83000, 2),
          (4, 'Janet', 'King', 95000, 2)]
```

- each table describes attributes about single entitiy
- why not putting all the data in one table?
  - you repeat information
    - department name and location is pelled out for each employee
  - managing the data is difficult
    - what if the marketing department changes its name to Brand Marketing
    - each row would require an update

1.2 Querying Multiple Tables Using JOIN

- when you join tables in a query the database connect rows in both tables wjere the columns you specified for the join have matching values
- the query results then include columns from both tables if you requested them as part of the query
- they are different types of JOIN

```
In [11]: %%sql

         SELECT
             *
         FROM
             employees
         JOIN
             departments
         ON
             employees.dept_id = departments.dept_id;

 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.


Out[11]: [(1, 'Nancy', 'Jones', 62500, 1, 1, 'Tax', 'Atalanta'),
          (2, 'Lee', 'Smith', 59300, 1, 1, 'Tax', 'Atalanta'),
          (3, 'Soo', 'Nguyen', 83000, 2, 2, 'IT', 'Boston'),
          (4, 'Janet', 'King', 95000, 2, 2, 'IT', 'Boston')]
```

- when you run the query the results include all values from both tables where values in the dept_id columns match
- in fact even the dept_id column appears twice because you selected all columns of both tables

1.3 JOIN Types

- there is more than one way to join tables in SQL
- it depends on how you want to retrive data
  - **JOIN** Returns every row from both tables
  - **LEFT JOIN** returns every row from the left table plus rows that match values in the joined column table from the right
  - **RIGHT JOIN**
  - **FULL OUTER JOIN** returns every row from both tables and matches rows, then joins the rows where values in the joined columns match
  - **CROSS JOIN** returns every possible cimbination of wors from both tables

1.3.1 Creating the Tables and fill it with Data

- lets assume we have 2 tables (left and right) with identical data
- one has 4 rows and one has 5 rows
- just 3 of the rows match with each other
- this is a common task
- we can analyse which data matches from bith tables with different JOIN queries

```
In [14]: %%sql

         CREATE TABLE
             schools_left (
             id integer CONSTRAINT left_id_key PRIMARY KEY,
             left_School varchar(30)
             );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[14]: []
```

```
In [15]: %%sql

         CREATE TABLE
             schools_right (
             id integer CONSTRAINT right_id_key PRIMARY KEY,
             right_School varchar(30)
             );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[15]: []

In [17]: %%sql

         INSERT INTO
             schools_left (
                 id, left_school)
         VALUES
             (1, 'Oak Street School'),
             (2, 'Roosevelt High School'),
             (5, 'Washington Middle School'),
             (6, 'Jefferson High School');

 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.


Out[17]: []

In [18]: %%sql

         INSERT INTO
             schools_right (
                 id, right_school)
         VALUES
             (1, 'Oak Street School'),
             (2, 'Roosevelt High School'),
             (3, 'Morrison Elementary'),
             (4, 'Chase Magnet Academy'),
             (6, 'Jefferson High School');

 * postgresql://postgres:***@localhost:5432/analysis
5 rows affected.


Out[18]: []

In [20]: %%sql

         SELECT * FROM schools_left;

 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.


Out[20]: [(1, 'Oak Street School'),
          (2, 'Roosevelt High School'),
          (5, 'Washington Middle School'),
          (6, 'Jefferson High School')]
```

```
In [21]: %%sql

         SELECT * FROM schools_right;

 * postgresql://postgres:***@localhost:5432/analysis
5 rows affected.


Out[21]: [(1, 'Oak Street School'),
          (2, 'Roosevelt High School'),
          (3, 'Morrison Elementary'),
          (4, 'Chase Magnet Academy'),
          (6, 'Jefferson High School')]
```

- we CREATE our two tables with the primary keys for each one
- for each table the id row must be filled
- we use INSERT statements to fill some data into the tables

1.3.2 JOIN

```
In [23]: %%sql

         SELECT
             *
         FROM
             schools_left
         JOIN
             schools_right
         ON
             schools_left.id = schools_right.id;

 * postgresql://postgres:***@localhost:5432/analysis
3 rows affected.


Out[23]: [(1, 'Oak Street School', 1, 'Oak Street School'),
          (2, 'Roosevelt High School', 2, 'Roosevelt High School'),
          (6, 'Jefferson High School', 6, 'Jefferson High School')]
```

- we use **JOIN** or **INNER JOIN** when we want to return rows that have a match in the columns we used for the join
- schools that exist only in one of two tables don't appear in the result
- when working with well structured, well mentained data sets and only need to find rows that exist in all the tables
- JOIN doesn't provide rows that exist in onyl one of the tables

1.3.3 LEFT JOIN and RIGHT JOIN

```
In [24]: %%sql
```

```
        SELECT
            *
        FROM
            schools_left LEFT JOIN schools_right
        ON
            schools_left.id = schools_right.id;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.
```

```
Out[24]: [(1, 'Oak Street School', 1, 'Oak Street School'),
          (2, 'Roosevelt High School', 2, 'Roosevelt High School'),
          (5, 'Washington Middle School', None, None),
          (6, 'Jefferson High School', 6, 'Jefferson High School')]
```

```
In [25]: %%sql

        SELECT
            *
        FROM
            schools_left RIGHT JOIN schools_right
        ON
            schools_left.id = schools_right.id;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
5 rows affected.
```

```
Out[25]: [(1, 'Oak Street School', 1, 'Oak Street School'),
          (2, 'Roosevelt High School', 2, 'Roosevelt High School'),
          (None, None, 3, 'Morrison Elementary'),
          (None, None, 4, 'Chase Magnet Academy'),
          (6, 'Jefferson High School', 6, 'Jefferson High School')]
```

- each return all rows from one table and display blank rows from the other table if no matching values are found in the joined columns
- this is good for:
  - you want to query results to contain all the rows from one of the tables
  - you want to look for missing values

1.3.4 FULL OUTER JOIN

```
In [26]: %%sql

        SELECT
            *
        FROM
            schools_left FULL OUTER JOIN schools_right
        On schools_left.id = schools_right.id;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
6 rows affected.
```

```
Out[26]: [(1, 'Oak Street School', 1, 'Oak Street School'),
          (2, 'Roosevelt High School', 2, 'Roosevelt High School'),
          (5, 'Washington Middle School', None, None),
          (6, 'Jefferson High School', 6, 'Jefferson High School'),
          (None, None, 4, 'Chase Magnet Academy'),
          (None, None, 3, 'Morrison Elementary')]
```

- when you want ot see al rows from both tables in a join
- regardless of whether any match
  - to merge two data sources that partically overlap or to visualize the degree to which the tables share matching values

1.3.5 CROSS JOIN

```
In [27]: %%sql

         SELECT
             *
         FROM
             schools_left CROSS JOIN schools_right;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
20 rows affected.
```

```
Out[27]: [(1, 'Oak Street School', 1, 'Oak Street School'),
          (1, 'Oak Street School', 2, 'Roosevelt High School'),
          (1, 'Oak Street School', 3, 'Morrison Elementary'),
          (1, 'Oak Street School', 4, 'Chase Magnet Academy'),
          (1, 'Oak Street School', 6, 'Jefferson High School'),
          (2, 'Roosevelt High School', 1, 'Oak Street School'),
          (2, 'Roosevelt High School', 2, 'Roosevelt High School'),
          (2, 'Roosevelt High School', 3, 'Morrison Elementary'),
          (2, 'Roosevelt High School', 4, 'Chase Magnet Academy'),
          (2, 'Roosevelt High School', 6, 'Jefferson High School'),
          (5, 'Washington Middle School', 1, 'Oak Street School'),
          (5, 'Washington Middle School', 2, 'Roosevelt High School'),
          (5, 'Washington Middle School', 3, 'Morrison Elementary'),
          (5, 'Washington Middle School', 4, 'Chase Magnet Academy'),
          (5, 'Washington Middle School', 6, 'Jefferson High School'),
          (6, 'Jefferson High School', 1, 'Oak Street School'),
          (6, 'Jefferson High School', 2, 'Roosevelt High School'),
          (6, 'Jefferson High School', 3, 'Morrison Elementary'),
          (6, 'Jefferson High School', 4, 'Chase Magnet Academy'),
          (6, 'Jefferson High School', 6, 'Jefferson High School')]
```

- the result of our 4 rows in one with the 5 rows in the other table equals 20 rows
- is the combination of all rows with each other

1.4 Using NULL to Find Rows with Missing Values

- in SQL **NULL** is a special value that represents a condition in which there's no data present or where the data is unknown because it wasn't included
- for example if a person filling out an adress skips the middle initial field rather than storing the unkwosn value
- it comes from **to nullify** and ahs nothing to do with **0**
- so you use NULL for unknown values

```
In [28]: %%sql

         SELECT
             *
         FROM
             schools_left LEFT JOIN schools_right
         ON
             schools_left.id = schools_right.id
         WHERE
             schools_right.id IS NULL;

 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.


Out[28]: [(5, 'Washington Middle School', None, None)]
```

- we want to look for columns *with no* data from the left table
- if we wanted to look for columns with data we would use **IS NOT NULL**
- the query just shows only the one rows from the left table that didn't have a match on the right side

1.5 Three Types of Table Relationships

1.5.1 One-to-One Relationship

- there is only one match for an id on each of the two tables
- there is no duplicate id values in either table

1.5.2 One-to-Many Relationship

- a key value in the first table will have multiple matching values in the second table's column
  - one table would hold data on manufacturers
  - the second table would hold the models for each manufacturer

1.5.3 Many-to-Many Relationship

- multiple rows in the first table will have multiple wors in the second table matching
  - baseball players could be joined to a table of field positions
  - each player can be assigned to multiple positions and each position can be played by multiple people

1.6 Selecting Specific Columns in a Join

- so far we used the asterisk wildcard to select all columns from both tables
- if you want to specify the query you have to specify the subset of columns
- you must give the desired columns name as well its table name

```
In [30]: %%sql

         SELECT
             id
         FROM
             schools_left LEFT JOIN schools_right
         ON
             schools_left.id = schools_right.id;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
(psycopg2.errors.AmbiguousColumn) column reference "id" is ambiguous
LINE 1: SELECT id
               ^

[SQL: SELECT id
FROM
    schools_left LEFT JOIN schools_right
ON
    schools_left.id = schools_right.id;]
(Background on this error at: http://sqlalche.me/e/f405)
```

- because **"id"** exist in both schools_left and schools_right the server gives an error
- it is not clear which table id belongs to

```
In [31]: %%sql

         SELECT
             schools_left.id,
             schools_left.left_school,
             schools_right.right_school
         FROM
             schools_left LEFT JOIN schools_right
         ON
             schools_left.id = schools_right.id;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.
```

```
Out[31]: [(1, 'Oak Street School', 'Oak Street School'),
          (2, 'Roosevelt High School', 'Roosevelt High School'),
          (5, 'Washington Middle School', None),
          (6, 'Jefferson High School', 'Jefferson High School')]
```

- we simply **prefix** each columns name with the table it comes from
- we can also add **AS** keyword to make it clear in the results that the id column is from schools_left
  - SELECT schools_left AS left_id ...

1.7 Simplifying JOIN Syntax with Table Aliases

```
In [32]: %%sql

         SELECT
             lt.id,
             lt.left_school,
             rt.right_School
         FROM
             schools_left AS lt LEFT JOIN schools_right AS rt
         ON
             lt.id = rt.id;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.
```

```
Out[32]: [(1, 'Oak Street School', 'Oak Street School'),
          (2, 'Roosevelt High School', 'Roosevelt High School'),
          (5, 'Washington Middle School', None),
          (6, 'Jefferson High School', 'Jefferson High School')]
```

- we simplify code with table aliases
- we do that in the **FROM** statement we declare here which aliases schould represent our tables using the **AS** keyword
- once declares we can use this everywhere in the code which makes the code more readable

1.8 Joining Multiple Tables

```
In [33]: %%sql

         CREATE TABLE
             schools_enrollment (
             id integer,
             enrollment integer
             );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

Out[33]: []

In [34]:
```
%%sql

CREATE TABLE
    schools_grades (
    id integer,
    grades varchar(10)
    );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

Out[34]: []

In [35]:
```
%%sql

INSERT INTO
    schools_enrollment (id, enrollment)
VALUES
    (1, 360),
    (2, 1001),
    (5, 450),
    (6, 927);
```

```
 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.
```

Out[35]: []

In [36]:
```
%%sql

INSERT INTO
    schools_grades (id, grades)
VALUES
    (1, 'K-3'),
    (2, '9-12'),
    (5, '6-8'),
    (6, '9-12');
```

```
 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.
```

Out[36]: []

In [39]: %%sql

```
SELECT lt.id, lt.left_school, en.enrollment, gr.grades
FROM schools_left AS lt LEFT JOIN schools_enrollment AS en
    ON lt.id = en.id
LEFT JOIN schools_grades AS gr
    ON lt.id = gr.id;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.
```

Out[39]:
```
[(1, 'Oak Street School', 360, 'K-3'),
 (2, 'Roosevelt High School', 1001, '9-12'),
 (5, 'Washington Middle School', 450, '6-8'),
 (6, 'Jefferson High School', 927, '9-12')]
```

- we create two more tables
- we insert data into the tables
- we use the select statement and give the columns we want to select
- we define our tables with the aliases
- we join them with the left join and give
- also we use the left join which corresponds to the from statement for the grades
- we now have three tables
- we can add more joins to have multiple tables added

1.9 Performing Math on Joined Table Columns

In [51]: %%sql

```
CREATE TABLE us_counties_2000 (
    geo_name varchar(90),              -- County/state name,
    state_us_abbreviation varchar(2),  -- State/U.S. abbreviation
    state_fips varchar(2),             -- State FIPS code
    county_fips varchar(3),            -- County code
    p0010001 integer,                  -- Total population
    p0010002 integer,                  -- Population of one race:
    p0010003 integer,                      -- White Alone
    p0010004 integer,                      -- Black or African American alone
    p0010005 integer,                      -- American Indian and Alaska Native alone
    p0010006 integer,                      -- Asian alone
    p0010007 integer,                      -- Native Hawaiian and Other Pacific Islan
    p0010008 integer,                      -- Some Other Race alone
    p0010009 integer,                  -- Population of two or more races
    p0010010 integer,                  -- Population of two races
    p0020002 integer,                  -- Hispanic or Latino
    p0020003 integer                   -- Not Hispanic or Latino:
);
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

Out[51]: []

In [53]:
```
%%sql

        COPY us_counties_2000
        FROM '/Users/ugurtigu/Documents/Learn/Docs/SQL/us_counties_2000.csv'
        WITH (FORMAT CSV, HEADER);
```

```
 * postgresql://postgres:***@localhost:5432/analysis
3141 rows affected.
```

Out[53]: []

In [56]:
```
%%sql

        SELECT c2010.geo_name,
               c2010.state_us_abbreviation AS state,
               c2010.p0010001 AS pop_2010,
               c2000.p0010001 AS pop_2000,
               c2010.p0010001 - c2000.p0010001 AS raw_change,
               round( (CAST(c2010.p0010001 AS numeric(8,1)) - c2000.p0010001)
                   / c2000.p0010001 * 100, 1 ) AS pct_change
        FROM us_counties_2010 AS c2010 INNER JOIN us_counties_2000 AS c2000
        ON c2010.state_fips = c2000.state_fips
           AND c2010.county_fips = c2000.county_fips
           AND c2010.p0010001 <> c2000.p0010001
        ORDER BY pct_change DESC
        LIMIT 100;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
100 rows affected.
```

Out[56]:
```
[('Kendall County', 'IL', 114736, 54544, 60192, Decimal('110.4')),
 ('Kendall County', 'IL', 114736, 54544, 60192, Decimal('110.4')),
 ('Pinal County', 'AZ', 375770, 179727, 196043, Decimal('109.1')),
 ('Pinal County', 'AZ', 375770, 179727, 196043, Decimal('109.1')),
 ('Flagler County', 'FL', 95696, 49832, 45864, Decimal('92.0')),
 ('Flagler County', 'FL', 95696, 49832, 45864, Decimal('92.0')),
 ('Lincoln County', 'SD', 44828, 24131, 20697, Decimal('85.8')),
 ('Lincoln County', 'SD', 44828, 24131, 20697, Decimal('85.8')),
 ('Loudoun County', 'VA', 312311, 169599, 142712, Decimal('84.1')),
 ('Loudoun County', 'VA', 312311, 169599, 142712, Decimal('84.1')),
 ('Rockwall County', 'TX', 78337, 43080, 35257, Decimal('81.8')),
```

```
('Rockwall County', 'TX', 78337, 43080, 35257, Decimal('81.8')),
('Forsyth County', 'GA', 175511, 98407, 77104, Decimal('78.4')),
('Forsyth County', 'GA', 175511, 98407, 77104, Decimal('78.4')),
('Sumter County', 'FL', 93420, 53345, 40075, Decimal('75.1')),
('Sumter County', 'FL', 93420, 53345, 40075, Decimal('75.1')),
('Paulding County', 'GA', 142324, 81678, 60646, Decimal('74.3')),
('Paulding County', 'GA', 142324, 81678, 60646, Decimal('74.3')),
('Sublette County', 'WY', 10247, 5920, 4327, Decimal('73.1')),
('Sublette County', 'WY', 10247, 5920, 4327, Decimal('73.1')),
('Henry County', 'GA', 203922, 119341, 84581, Decimal('70.9')),
('Henry County', 'GA', 203922, 119341, 84581, Decimal('70.9')),
('Teton County', 'ID', 10170, 5999, 4171, Decimal('69.5')),
('Teton County', 'ID', 10170, 5999, 4171, Decimal('69.5')),
('Williamson County', 'TX', 422679, 249967, 172712, Decimal('69.1')),
('Williamson County', 'TX', 422679, 249967, 172712, Decimal('69.1')),
('Fort Bend County', 'TX', 585375, 354452, 230923, Decimal('65.1')),
('Fort Bend County', 'TX', 585375, 354452, 230923, Decimal('65.1')),
('Union County', 'NC', 201292, 123677, 77615, Decimal('62.8')),
('Union County', 'NC', 201292, 123677, 77615, Decimal('62.8')),
('Douglas County', 'CO', 285465, 175766, 109699, Decimal('62.4')),
('Douglas County', 'CO', 285465, 175766, 109699, Decimal('62.4')),
('Dallas County', 'IA', 66135, 40750, 25385, Decimal('62.3')),
('Dallas County', 'IA', 66135, 40750, 25385, Decimal('62.3')),
('Newton County', 'GA', 99958, 62001, 37957, Decimal('61.2')),
('Newton County', 'GA', 99958, 62001, 37957, Decimal('61.2')),
('Hays County', 'TX', 157107, 97589, 59518, Decimal('61.0')),
('Hays County', 'TX', 157107, 97589, 59518, Decimal('61.0')),
('Collin County', 'TX', 782341, 491675, 290666, Decimal('59.1')),
('Collin County', 'TX', 782341, 491675, 290666, Decimal('59.1')),
('Delaware County', 'OH', 174214, 109989, 64225, Decimal('58.4')),
('Franklin County', 'WA', 78163, 49347, 28816, Decimal('58.4')),
('Delaware County', 'OH', 174214, 109989, 64225, Decimal('58.4')),
('Franklin County', 'WA', 78163, 49347, 28816, Decimal('58.4')),
('Forest County', 'PA', 7716, 4946, 2770, Decimal('56.0')),
('Forest County', 'PA', 7716, 4946, 2770, Decimal('56.0')),
('Osceola County', 'FL', 268685, 172493, 96192, Decimal('55.8')),
('Osceola County', 'FL', 268685, 172493, 96192, Decimal('55.8')),
('Montgomery County', 'TX', 455746, 293768, 161978, Decimal('55.1')),
('Montgomery County', 'TX', 455746, 293768, 161978, Decimal('55.1')),
('Wasatch County', 'UT', 23530, 15215, 8315, Decimal('54.7')),
('Wasatch County', 'UT', 23530, 15215, 8315, Decimal('54.7')),
('St. Johns County', 'FL', 190039, 123135, 66904, Decimal('54.3')),
('St. Johns County', 'FL', 190039, 123135, 66904, Decimal('54.3')),
('Denton County', 'TX', 662614, 432976, 229638, Decimal('53.0')),
('Denton County', 'TX', 662614, 432976, 229638, Decimal('53.0')),
('Washington County', 'UT', 138115, 90354, 47761, Decimal('52.9')),
('Washington County', 'UT', 138115, 90354, 47761, Decimal('52.9')),
('Cherokee County', 'GA', 214346, 141903, 72443, Decimal('51.1')),
```

```
('Cherokee County', 'GA', 214346, 141903, 72443, Decimal('51.1')),
('Lyon County', 'NV', 51980, 34501, 17479, Decimal('50.7')),
('Lyon County', 'NV', 51980, 34501, 17479, Decimal('50.7')),
('DeSoto County', 'MS', 161252, 107199, 54053, Decimal('50.4')),
('DeSoto County', 'MS', 161252, 107199, 54053, Decimal('50.4')),
('Hamilton County', 'IN', 274569, 182740, 91829, Decimal('50.3')),
('Hamilton County', 'IN', 274569, 182740, 91829, Decimal('50.3')),
('Barrow County', 'GA', 69367, 46144, 23223, Decimal('50.3')),
('Barrow County', 'GA', 69367, 46144, 23223, Decimal('50.3')),
('Matanuska-Susitna Borough', 'AK', 88995, 59322, 29673, Decimal('50.0')),
('Matanuska-Susitna Borough', 'AK', 88995, 59322, 29673, Decimal('50.0')),
('Guadalupe County', 'TX', 131533, 89023, 42510, Decimal('47.8')),
('Guadalupe County', 'TX', 131533, 89023, 42510, Decimal('47.8')),
('Brunswick County', 'NC', 107431, 73143, 34288, Decimal('46.9')),
('Brunswick County', 'NC', 107431, 73143, 34288, Decimal('46.9')),
('Sandoval County', 'NM', 131561, 89908, 41653, Decimal('46.3')),
('Sandoval County', 'NM', 131561, 89908, 41653, Decimal('46.3')),
('Jackson County', 'GA', 60485, 41589, 18896, Decimal('45.4')),
('Jackson County', 'GA', 60485, 41589, 18896, Decimal('45.4')),
('Scott County', 'MN', 129928, 89498, 40430, Decimal('45.2')),
('Scott County', 'MN', 129928, 89498, 40430, Decimal('45.2')),
('Spencer County', 'KY', 17061, 11766, 5295, Decimal('45.0')),
('Spencer County', 'KY', 17061, 11766, 5295, Decimal('45.0')),
('Camden County', 'NC', 9980, 6885, 3095, Decimal('45.0')),
('Camden County', 'NC', 9980, 6885, 3095, Decimal('45.0')),
('Kaufman County', 'TX', 103350, 71313, 32037, Decimal('44.9')),
('Kaufman County', 'TX', 103350, 71313, 32037, Decimal('44.9')),
('Williamson County', 'TN', 183182, 126638, 56544, Decimal('44.7')),
('Williamson County', 'TN', 183182, 126638, 56544, Decimal('44.7')),
('Benton County', 'AR', 221339, 153406, 67933, Decimal('44.3')),
('Benton County', 'AR', 221339, 153406, 67933, Decimal('44.3')),
('Rutherford County', 'TN', 262604, 182023, 80581, Decimal('44.3')),
('Rutherford County', 'TN', 262604, 182023, 80581, Decimal('44.3')),
('St. Lucie County', 'FL', 277789, 192695, 85094, Decimal('44.2')),
('St. Lucie County', 'FL', 277789, 192695, 85094, Decimal('44.2')),
('Canyon County', 'ID', 188923, 131441, 57482, Decimal('43.7')),
('Canyon County', 'ID', 188923, 131441, 57482, Decimal('43.7')),
('Douglas County', 'GA', 132403, 92174, 40229, Decimal('43.6')),
('Douglas County', 'GA', 132403, 92174, 40229, Decimal('43.6')),
('Wake County', 'NC', 900993, 627846, 273147, Decimal('43.5')),
('Wake County', 'NC', 900993, 627846, 273147, Decimal('43.5'))]
```

- we first create a table with the needed fields
- we then copy our data source into us_counties_2000 (which is another file than 2010)
- we calculate the population value from 2000 and 2010 with the percentage change
- we calculate the raw change with the minus operation one line before
- we join by matching values in two columns on both tables
  - state_fips

  - county_fips

- we have another condition to illustrate inequality
- we combine conditions with the **AND** keyword
- all three conditions has to be satisfied to make the JOIN possible
- we now can see topp 100 counties where the population grows most

1.9.1 Tasks

– 1. The table us_counties_2010 contains 3,143 rows, and us_counties_2000 has – 3,141. That reflects the ongoing adjustments to county-level geographies that – typically result from government decision making. Using appropriate joins and – the NULL value, identify which counties don't exist in both tables.

In [64]:
```
%%sql

        SELECT c2010.geo_name,
               c2010.state_us_abbreviation,
               c2000.geo_name
        FROM us_counties_2010 c2010 LEFT JOIN us_counties_2000 c2000
        ON c2010.state_fips = c2000.state_fips
           AND c2010.county_fips = c2000.county_fips
        WHERE c2000.geo_name IS NULL;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
12 rows affected.
```

Out[64]:
```
[('Hoonah-Angoon Census Area', 'AK', None),
 ('Hoonah-Angoon Census Area', 'AK', None),
 ('Petersburg Census Area', 'AK', None),
 ('Petersburg Census Area', 'AK', None),
 ('Prince of Wales-Hyder Census Area', 'AK', None),
 ('Prince of Wales-Hyder Census Area', 'AK', None),
 ('Skagway Municipality', 'AK', None),
 ('Skagway Municipality', 'AK', None),
 ('Wrangell City and Borough', 'AK', None),
 ('Wrangell City and Borough', 'AK', None),
 ('Broomfield County', 'CO', None),
 ('Broomfield County', 'CO', None)]
```

- we want to check which new rows are added to the 2010 table
- therefore we have to check which entries are missing (NULL) we have to nullify them from the old list
- the 2010 table contains 3143 entries and the 2000 just 3141

In [65]:
```
%%sql

        SELECT c2010.geo_name,
```

```
            c2000.geo_name,
            c2000.state_us_abbreviation
        FROM us_counties_2010 c2010 RIGHT JOIN us_counties_2000 c2000
        ON c2010.state_fips = c2000.state_fips
           AND c2010.county_fips = c2000.county_fips
        WHERE c2010.geo_name IS NULL;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
4 rows affected.
```

```
Out[65]: [(None, 'Prince of Wales-Outer Ketchikan Census Area, Alaska', 'AK'),
          (None, 'Skagway-Hoonah-Angoon Census Area, Alaska', 'AK'),
          (None, 'Wrangell-Petersburg Census Area, Alaska', 'AK'),
          (None, 'Clifton Forge city, Virginia', 'VA')]
```

- Counties that exist in 2000 data but not 2010

– 2. Using either the median() or percentile_cont() functions in Chapter 5, – determine the median of the percent change in county population

```
In [67]: %%sql

        SELECT median(round( (CAST(c2010.p0010001 AS numeric(8,1))
                              - c2000.p0010001)
                    / c2000.p0010001 * 100, 1 )) AS median_pct_change
        FROM us_counties_2010 c2010 JOIN us_counties_2000 c2000
        ON c2010.state_fips = c2000.state_fips
           AND c2010.county_fips = c2000.county_fips;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.
```

```
Out[67]: [(3.2,)]
```

- we first use the select operation
- then we do our math operation
- here we use the median function we created earlier
- then we use the round function and cast the population of each year with a numeric number
- we take the population of the year 2010 minus the population of the year 2000 and devide it with the 2000 value to see the percentual change
- from this value we take the meduam after we round it

– 3. Which county had the greatest percentage loss of population between 2000 – and 2010? Do you have any idea why? Hint: a weather event happened in 2005.

```
In [69]: %%sql
```

```
        SELECT c2010.geo_name,
               c2010.state_us_abbreviation,
               c2010.p0010001 AS pop_2010,
               c2000.p0010001 AS pop_2000,
               c2010.p0010001 - c2000.p0010001 AS raw_change,
               round( (CAST(c2010.p0010001 AS DECIMAL(8,1)) - c2000.p0010001)
                   / c2000.p0010001 * 100, 1 ) AS pct_change
        FROM us_counties_2010 c2010 INNER JOIN us_counties_2000 c2000
        ON c2010.state_fips = c2000.state_fips
           AND c2010.county_fips = c2000.county_fips
        ORDER BY pct_change ASC
        LIMIT 10;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
10 rows affected.
```

Out[69]:
```
[('St. Bernard Parish', 'LA', 35897, 67229, -31332, Decimal('-46.6')),
 ('St. Bernard Parish', 'LA', 35897, 67229, -31332, Decimal('-46.6')),
 ('Kalawao County', 'HI', 90, 147, -57, Decimal('-38.8')),
 ('Kalawao County', 'HI', 90, 147, -57, Decimal('-38.8')),
 ('Issaquena County', 'MS', 1406, 2274, -868, Decimal('-38.2')),
 ('Issaquena County', 'MS', 1406, 2274, -868, Decimal('-38.2')),
 ('Cameron Parish', 'LA', 6839, 9991, -3152, Decimal('-31.5')),
 ('Cameron Parish', 'LA', 6839, 9991, -3152, Decimal('-31.5')),
 ('Orleans Parish', 'LA', 343829, 484674, -140845, Decimal('-29.1')),
 ('Orleans Parish', 'LA', 343829, 484674, -140845, Decimal('-29.1'))]
```

In [ ]: