# Chapter 15

November 16, 2019

[1]:
```
%load_ext sql
```

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

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

## 1 Saving Time with Views, Functions, and Triggers

### 1.1 Creating and Querying Views

[3]:
```
%%sql

CREATE OR REPLACE VIEW nevada_counties_pop_2010 AS
    SELECT geo_name,
            state_fips,
            county_fips,
            p0010001 AS pop_2010
    FROM us_counties_2010
    WHERE state_us_abbreviation = 'NV'
    ORDER BY county_fips;
```

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

[3]:
```
[]
```

- we define the view using the keyword **CREATE OR REPLACE VIEW** followed by the name of the view with the **AS** keyword
- we do a select query for the county nevada (NV)
- then we order the data by the countie's FIPS
- the **OR REPLACE** after the **CREATE** in the first line has to be noticed
  - if its already exists, create it

[4]:
```
%%sql

SELECT *
FROM nevada_counties_pop_2010
```

```
LIMIT 5;
```

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

[4]:
```
[('Churchill County', '32', '001', 24877),
 ('Clark County', '32', '003', 1951269),
 ('Douglas County', '32', '005', 46997),
 ('Elko County', '32', '007', 48818),
 ('Esmeralda County', '32', '009', 783)]
```

- like a ordinary table we can select and use the view

[5]:
```
%%sql

SELECT geo_name,
        state_fips,
        county_fips,
        p0010001 AS pop_2010
FROM us_counties_2010
WHERE state_us_abbreviation = 'NV'
ORDER BY county_fips
LIMIT 5;
```

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

[5]:
```
[('Churchill County', '32', '001', 24877),
 ('Clark County', '32', '003', 1951269),
 ('Douglas County', '32', '005', 46997),
 ('Elko County', '32', '007', 48818),
 ('Esmeralda County', '32', '009', 783)]
```

- see, it's the same as the query with the select statement from creation of the view
- if you do this task frequently (see the population of nevada) this view can be useful

[6]:
```
%%sql

CREATE OR REPLACE VIEW county_pop_change_2010_2000 AS
    SELECT c2010.geo_name,
            c2010.state_us_abbreviation AS st,
            c2010.state_fips,
            c2010.county_fips,
            c2010.p0010001 AS pop_2010,
            c2000.p0010001 AS pop_2000,
            round((CAST(c2010.p0010001 AS numeric(8,1)) - c2000.p0010001)
                 / c2000.p0010001 * 100, 1) AS pct_change_2010_2000
    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
    ORDER BY c2010.state_fips, c2010.county_fips;
```

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

[6]: []

- we create our view for calculationg the percentage change of the counties population and states by joining them together

[9]:
```
%%sql

SELECT geo_name,
        st,
        pop_2010,
        pop_2000,
        pct_change_2010_2000
FROM county_pop_change_2010_2000
WHERE st = 'NV'
LIMIT 5;
```

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

[9]:
```
[('Churchill County', 'NV', 24877, 23982, Decimal('3.7')),
 ('Clark County', 'NV', 1951269, 1375765, Decimal('41.8')),
 ('Douglas County', 'NV', 46997, 41259, Decimal('13.9')),
 ('Elko County', 'NV', 48818, 45291, Decimal('7.8')),
 ('Esmeralda County', 'NV', 783, 971, Decimal('-19.4'))]
```

- now that we created a view, we can use the code to run a simple query for the state nevada NV
- note that we can use sepcific columns when querying a view
- we are also filtering same as before with the where clause

### 1.2 Inserting, Updating, and Deleting Data Using a View

[15]:
```
%%sql

SELECT *
FROM employees;
```

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

[15]:
```
[(1, 'Nancy', 'Jones', 62500, 1),
 (2, 'Lee', 'Smith', 59300, 1),
 (3, 'Soo', 'Nguyen', 83000, 2),
 (4, 'Janet', 'King', 95000, 2)]
```

[13]:
```
%%sql

CREATE OR REPLACE VIEW employees_tax_dept AS
    SELECT emp_id,
            first_name,
            last_name,
            dept_id
    FROM employees
    WHERE dept_id = 1
    ORDER BY emp_id
    WITH LOCAL CHECK OPTION;
```

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

[13]:
```
[]
```

- we make a view for the department 1
- to restrict inserts or updated to this department employees only we use the last statement **WITH LOCAL CHECK OPTION** which rejects any insert or update that does not meet the criteria of the WHERE clause, for example WHERE dept_id = 3 wouldn't do any insert or update

[17]:
```
%%sql

SELECT *
FROM employees_tax_dept;
```

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

[17]:
```
[(1, 'Nancy', 'Jones', 1), (2, 'Lee', 'Smith', 1)]
```

### 1.2.1 Inserting Rows Using the employees_tax_dept View

[23]:
```
%%sql

INSERT INTO employees_tax_dept (first_name, last_name, dept_id)
VALUES ('Suzanne', 'Legere', 1);
```

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

[23]: []

- this query runs okay because we add a value with dept_id 1

[27]:
```
%%sql

INSERT INTO employees_tax_dept (first_name, last_name, dept_id)
VALUES ('Jamil', 'White', 2);
```

```
 * postgresql://postgres:***@localhost:5432/analysis
(psycopg2.errors.WithCheckOptionViolation) new row violates check option for
view "employees_tax_dept"
DETAIL:  Failing row contains (9, Jamil, White, null, 2).

[SQL: INSERT INTO employees_tax_dept (first_name, last_name, dept_id)
VALUES ('Jamil', 'White', 2);]
(Background on this error at: http://sqlalche.me/e/f405)
```

- however this query doesn't run okay because we are trying to add a value with the dept_id 2

[28]:
```
%%sql

SELECT *
FROM employees_tax_dept;
```

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

[28]:
```
[(1, 'Nancy', 'Jones', 1), (2, 'Lee', 'Smith', 1), (5, 'Suzanne', 'Legere', 1)]
```

[29]:
```
%%sql

SELECT *
FROM employees;
```

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

[29]:
```
[(1, 'Nancy', 'Jones', 62500, 1),
 (2, 'Lee', 'Smith', 59300, 1),
 (3, 'Soo', 'Nguyen', 83000, 2),
 (4, 'Janet', 'King', 95000, 2),
 (5, 'Suzanne', 'Legere', None, 1)]
```

- as you can see the data we add using the view is also addes to the underlying table
- however because the view doesn't have a **salary** column, we get a null value for this
- the view has no reference to salary

### 1.2.2 Updating Rows Using the employees_tax_dept View

[30]:
```
%%sql

UPDATE employees_tax_dept
SET last_name = 'Le Gere'
WHERE emp_id = 5;
```

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

[30]:
```
[]
```

[31]:
```
%%sql

SELECT *
FROM employees_tax_dept;
```

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

[31]:
```
[(1, 'Nancy', 'Jones', 1),
 (2, 'Lee', 'Smith', 1),
 (5, 'Suzanne', 'Le Gere', 1)]
```

- we can use the view to update the table

[32]:
```
%%sql

SELECT *
FROM employees;
```

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

[32]:
```
[(1, 'Nancy', 'Jones', 62500, 1),
 (2, 'Lee', 'Smith', 59300, 1),
 (3, 'Soo', 'Nguyen', 83000, 2),
 (4, 'Janet', 'King', 95000, 2),
 (5, 'Suzanne', 'Le Gere', None, 1)]
```

- however we can update just the dept_id 1

### 1.2.3 Deleting Rows Using the emlpyees_tax_dept View

[37]:
```
%%sql

DELETE FROM employees_tax_dept
WHERE emp_id = 5;
```

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

[37]: []

[34]:
```
%%sql

SELECT *
FROM employees_tax_dept;
```

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

[34]: [(1, 'Nancy', 'Jones', 1), (2, 'Lee', 'Smith', 1)]

[38]:
```
%%sql

DELETE FROM employees_tax_dept
WHERE emp_id = 2;
```

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

[38]: []

## 1.3 Programming Your Own Functions

### 1.3.1 Creating the percent_change() Function

- percent change = ( New Number - Old NUmber ) / Old Number

[41]:
```
%%sql

CREATE OR REPLACE FUNCTION
percent_change(new_value numeric,
               old_value numeric,
               decimal_places integer DEFAULT 1)
RETURNS numeric AS
'SELECT round(
        ((new_value - old_value) / old_value) * 100, decimal_places
);'
LANGUAGE SQL
IMMUTABLE
```

```
RETURNS NULL ON NULL INPUT;
```

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

[41]:
```
[]
```

- we start wirh the command CREATE OR REPLACE FUNCTION followed by the name of the function
- in the parantheses we add the arguments of the function and each arguments data type
  - we specify the new_value and old_value as numeric
  - the decimal_places (which specifies the number of places to round the results, is integer
  - we specify 1 as the DEFAULT, the argument will be optional when we call the function later
- we then use the keywords RETURNS numeric AS to tell the function to return numeric
- if this would be a concatenating function we would return text
- next we write the operation of the function
- inside the single quotes we do a SELCT operation with the mathematical things to do
- that includes a percentage change with the round function
- we used the argument instead of 1
- the languguage keyword we give
- and we say that the function will not make any changes to the database
- the last kewword guarantees that this function will supply a NULL response if any input is NULL

### 1.3.2 Using the percent_change() Function

[42]:
```
%%sql

SELECT percent_change(110, 108, 2);
```

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

[42]:
```
[(Decimal('1.85'),)]
```

[43]:
```
%%sql

SELECT percent_change(110, 108);
```

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

[43]:
```
[(Decimal('1.9'),)]
```

- note the difference bewteen both, the last one is without the decimal argument, here it will be by default 1

[44]:
```
%%sql

SELECT c2010.geo_name,
        c2010.state_us_abbreviation AS st,
        c2010.p0010001 AS pop_2010,
        percent_change(c2010.p0010001, c2000.p0010001) AS pct_chg_func,
        round( (CAST(c2010.p0010001 AS numeric(8,1)) -c2000.p0010001)
             / c2000.p0010001 * 100, 1 ) AS pct_chg_formula
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
ORDER BY pct_chg_func DESC
LIMIT 5;
```

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

[44]:
```
[('Kendall County', 'IL', 114736, Decimal('110.4'), Decimal('110.4')),
 ('Pinal County', 'AZ', 375770, Decimal('109.1'), Decimal('109.1')),
 ('Flagler County', 'FL', 95696, Decimal('92.0'), Decimal('92.0')),
 ('Lincoln County', 'SD', 44828, Decimal('85.8'), Decimal('85.8')),
 ('Loudoun County', 'VA', 312311, Decimal('84.1'), Decimal('84.1'))]
```

- here we can compare the results
- the first with the function
- the seond with the query

[50]:
```
%%sql

SELECT c2010.geo_name,
        c2010.state_us_abbreviation AS st,
        c2010.p0010001 AS pop_2010,
        percent_change(c2010.p0010001, c2000.p0010001) AS 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
ORDER BY change DESC
LIMIT 5;
```

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

[50]:
```
[('Kendall County', 'IL', 114736, Decimal('110.4')),
 ('Pinal County', 'AZ', 375770, Decimal('109.1')),
 ('Flagler County', 'FL', 95696, Decimal('92.0')),
 ('Lincoln County', 'SD', 44828, Decimal('85.8')),
 ('Loudoun County', 'VA', 312311, Decimal('84.1'))]
```

- we could do this
- we can use percentage change function any time we need to solve that calculation