Chapter 2

October 27, 2019

1 2. Beginning Data Exploration with Select

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

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

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

1.1 Basic SELECT Syntax

- SELECT * FROM my_table;
- this single line of code we select all columns of the table (note that the asterisk is a wildcard
- the FROM keyword indicates you want to query a particular table

```
In [4]: %%sql

        SELECT * FROM teachers;

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

```
Out[4]: [(1, 'Janet', 'Smith', 'F.D. Roosevelt HS', datetime.date(2011, 10, 30), Decimal('3620
         (2, 'Lee', 'Reynolds', 'F.D. Roosevelt HS', datetime.date(1993, 5, 22), Decimal('6500
         (3, 'Samuel', 'Cole', 'Myers Middle School', datetime.date(2005, 8, 1), Decimal('4350
         (4, 'Samantha', 'Bush', 'Myers Middle School', datetime.date(2011, 10, 30), Decimal('
         (5, 'Betty', 'Diaz', 'Myers Middle School', datetime.date(2005, 8, 30), Decimal('4350
         (6, 'Kathleen', 'Roush', 'F.D. Roosevelt HS', datetime.date(2010, 10, 22), Decimal('38
```

- we now see all rows of the table we selected
- note that the id column of the type bigserial automatically increments

1.2 Querying a subset of columns

- we can also just select a **subset of columns** by doing this: *SELECT some_column, another_column, amazing_column FROM table_name;

```
In [6]: %%sql

        SELECT last_name, first_name, salary FROM teachers;

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


Out[6]: [('Smith', 'Janet', Decimal('36200')),
         ('Reynolds', 'Lee', Decimal('65000')),
         ('Cole', 'Samuel', Decimal('43500')),
         ('Bush', 'Samantha', Decimal('36200')),
         ('Diaz', 'Betty', Decimal('43500')),
         ('Roush', 'Kathleen', Decimal('38500'))]
```

- now we selected just a subset of columns from the table
- note that the order of the table is now different than before
- you are able to retrieve columns in any way

- generally it is wise to check the data first
- so you can see the format of the dates and the input
- now we just have a table with 6 rows
- later if we have a table with million rows it is essential to get a quick read on your data

1.3 Using DISTINCT to find Unique Values

```
In [7]: %%sql

        SELECT DISTINCT school
        FROM teachers;

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


Out[7]: [('Myers Middle School',), ('F.D. Roosevelt HS',)]
```

- we can find duplicates from a column
- in the teachers table the column school lists the same school names multiple times
- we can so detect if a school name is written corectly or not by checking the variations of the names
- when you are working with dates discinct will help highlighting inconsistent or broken formatting

```
In [9]: %%sql

        SELECT DISTINCT school, salary
        FROM teachers;
```

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

```
Out[9]: [('Myers Middle School', Decimal('36200')),
         ('F.D. Roosevelt HS', Decimal('65000')),
         ('Myers Middle School', Decimal('43500')),
         ('F.D. Roosevelt HS', Decimal('38500')),
         ('F.D. Roosevelt HS', Decimal('36200'))]
```

- the DISTINCT keywoard also works with multiple columns
- because two teachers at MMSchool earn 43.500$ that pair is listed in just one row
- this is a way to query "For each x in the table, what are all the y values?"

1.4 Sorting Data with ORDER BY

```
In [10]: %%sql

         SELECT first_name, last_name, salary
         FROM teachers
         ORDER BY salary DESC;
```

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

```
Out[10]: [('Lee', 'Reynolds', Decimal('65000')),
          ('Samuel', 'Cole', Decimal('43500')),
          ('Betty', 'Diaz', Decimal('43500')),
          ('Kathleen', 'Roush', Decimal('38500')),
          ('Janet', 'Smith', Decimal('36200')),
          ('Samantha', 'Bush', Decimal('36200'))]
```

- ORDER BY sorts values in ascending order
- here we adding the keyword DESC to do the opposite
- we order by salary column from highest to lowsest

```
In [11]: %%sql

         SELECT last_name, school, hire_date
         FROM teachers
         ORDER BY school ASC, hire_date DESC;
```

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

```
Out[11]: [('Smith', 'F.D. Roosevelt HS', datetime.date(2011, 10, 30)),
          ('Roush', 'F.D. Roosevelt HS', datetime.date(2010, 10, 22)),
```

```
('Reynolds', 'F.D. Roosevelt HS', datetime.date(1993, 5, 22)),
('Bush', 'Myers Middle School', datetime.date(2011, 10, 30)),
('Diaz', 'Myers Middle School', datetime.date(2005, 8, 30)),
('Cole', 'Myers Middle School', datetime.date(2005, 8, 1))]
```

- note that we can also make queries which can sort not just one column
- we get a listing of teachers grouped by school with the most recently hired teachers first
- in other words, we see the newest teachers from each school

1.5 Filtering Rows with WHERE

```
In [12]: %%sql

         SELECT last_name, school, hire_date
         FROM teachers
         WHERE school = 'Myers Middle School'

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


Out[12]: [('Cole', 'Myers Middle School', datetime.date(2005, 8, 1)),
          ('Bush', 'Myers Middle School', datetime.date(2011, 10, 30)),
          ('Diaz', 'Myers Middle School', datetime.date(2005, 8, 30))]
```

- here we are using the = operator to find rows that *exactly* match a value
- you can use other operators
- with the WHERE keyword we say, that we want all the rows with the given value from our column school
- other operators such as
  - "!=" (not equal to)
  - ">" greater than
  - "<" less than
  - "LIKE" match a case sensitive pattern (LIKE 'Sam%')
  - "ILIKE" match a case sensitive pattern (ILIKE '%sam')

```
In [13]: %%sql

         SELECT first_name, last_name, school
         FROM teachers
         WHERE first_name = 'Janet';

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


Out[13]: [('Janet', 'Smith', 'F.D. Roosevelt HS')]
```

```
In [17]: %%sql

         SELECT school
         FROM teachers
         WHERE school != 'F.D. Roosevelt HS';

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


Out[17]: [('Myers Middle School',), ('Myers Middle School',), ('Myers Middle School',)]

In [18]: %%sql

         SELECT first_name, last_name, hire_date
         FROM teachers
         WHERE hire_date < '2000-01-01';

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


Out[18]: [('Lee', 'Reynolds', datetime.date(1993, 5, 22))]

In [15]: %%sql

         SELECT first_name, last_name, school
         FROM teachers
         WHERE salary >= 43500;

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


Out[15]: [('Lee', 'Reynolds', 'F.D. Roosevelt HS'),
          ('Samuel', 'Cole', 'Myers Middle School'),
          ('Betty', 'Diaz', 'Myers Middle School')]

In [16]: %%sql

         SELECT first_name, last_name, school
         FROM teachers
         WHERE salary BETWEEN 40000 AND 65000;

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


Out[16]: [('Lee', 'Reynolds', 'F.D. Roosevelt HS'),
          ('Samuel', 'Cole', 'Myers Middle School'),
          ('Betty', 'Diaz', 'Myers Middle School')]
```

1.6 Using LIKE and ILIKE with WHERE

- let you search for pattersn in strings by using two special characters
  - percent sign (%) - a wildcard matching one or more characters
  - underscore (_) - a wildcard matching just one character
  - for example if you want to find the word *baker* the following LIKE patterns will match it
    - LIKE 'b%'
    - LIKE '%ak%'
    - LIKE '_aker'
    - LIke 'ba_er'

```
In [25]: %%sql

         SELECT first_name
         FROM teachers
         WHERE first_name LIKE 'sam%';

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


Out[25]: []
```

```
In [23]: %%sql

         SELECT first_name
         FROM teachers
         WHERE first_name ILIKE 'sam%';

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


Out[23]: [('Samuel',), ('Samantha',)]
```

- you can see the difference between LIKE and ILIKE better now
- ILIKE is not case sensitive
- it is a PostgreSQL-only implementation

1.7 Combining Operators with AND and OR

```
In [26]: %%sql

         SELECT *
         FROM teachers
         WHERE school = 'Myers Middle School'
                 AND salary < 40000;
```

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

```
Out[26]: [(4, 'Samantha', 'Bush', 'Myers Middle School', datetime.date(2011, 10, 30), Decimal(
```

```
In [27]: %%sql

         SELECT *
         FROM teachers
         WHERE last_name = 'Cole'
                 OR last_name = 'Bush';
```

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

```
Out[27]: [(3, 'Samuel', 'Cole', 'Myers Middle School', datetime.date(2005, 8, 1), Decimal('435
          (4, 'Samantha', 'Bush', 'Myers Middle School', datetime.date(2011, 10, 30), Decimal(
```

```
In [28]: %%sql

         SELECT *
         FROM teachers
         WHERE school = 'F.D. Roosevelt HS'
                 AND (salary < 38000 OR salary > 40000);
```

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

```
Out[28]: [(1, 'Janet', 'Smith', 'F.D. Roosevelt HS', datetime.date(2011, 10, 30), Decimal('362
          (2, 'Lee', 'Reynolds', 'F.D. Roosevelt HS', datetime.date(1993, 5, 22), Decimal('650
```

- we combining where clauses with AND and OR keywords
- we also combining opertors with AND and OR keywords

1.8 Putting It All Together

- you can combine filtering with AND and OR into one statemen
- SQL is particular about the order of keywords
  - SELECT column_names
  - FROM table_name
  - WHERE criteria
  - ORDER BY column_names

```
In [31]: %%sql

         SELECT first_name, last_name, school, hire_date, salary
         FROM teachers
         WHERE school LIKE '%Roos%'
         ORDER BY hire_date DESC;
```

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

```
Out[31]: [('Janet', 'Smith', 'F.D. Roosevelt HS', datetime.date(2011, 10, 30), Decimal('36200')
          ('Kathleen', 'Roush', 'F.D. Roosevelt HS', datetime.date(2010, 10, 22), Decimal('385
          ('Lee', 'Reynolds', 'F.D. Roosevelt HS', datetime.date(1993, 5, 22), Decimal('65000')
```

1.9 Try It Yourself

– 1. The school district superintendent asks for a list of teachers in each – school. Write a query that lists the schools in alphabetical order along – with teachers ordered by last name A-Z.

```
In [34]: %%sql

         SELECT first_name, last_name, school
         FROM teachers
         ORDER BY last_name ASC;
```

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

```
Out[34]: [('Samantha', 'Bush', 'Myers Middle School'),
          ('Samuel', 'Cole', 'Myers Middle School'),
          ('Betty', 'Diaz', 'Myers Middle School'),
          ('Lee', 'Reynolds', 'F.D. Roosevelt HS'),
          ('Kathleen', 'Roush', 'F.D. Roosevelt HS'),
          ('Janet', 'Smith', 'F.D. Roosevelt HS')]
```

– 2. Write a query that finds the one teacher whose first name starts – with the letter 'S' and who earns more than $40,000.

```
In [36]: %%sql

         SELECT first_name, salary
         FROM teachers
         WHERE first_name LIKE 'S%'
         AND salary >= 40000;
```

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

```
Out[36]: [('Samuel', Decimal('43500'))]
```

– 3. Rank teachers hired since Jan. 1, 2010, ordered by highest paid to lowest.

In [42]:
```
%%sql

        SELECT first_name, last_name, hire_date, salary
        FROM teachers
        WHERE hire_date >= '2010-01-01'
        ORDER BY salary DESC;
```

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

Out[42]:
```
[('Kathleen', 'Roush', datetime.date(2010, 10, 22), Decimal('38500')),
 ('Janet', 'Smith', datetime.date(2011, 10, 30), Decimal('36200')),
 ('Samantha', 'Bush', datetime.date(2011, 10, 30), Decimal('36200'))]
```