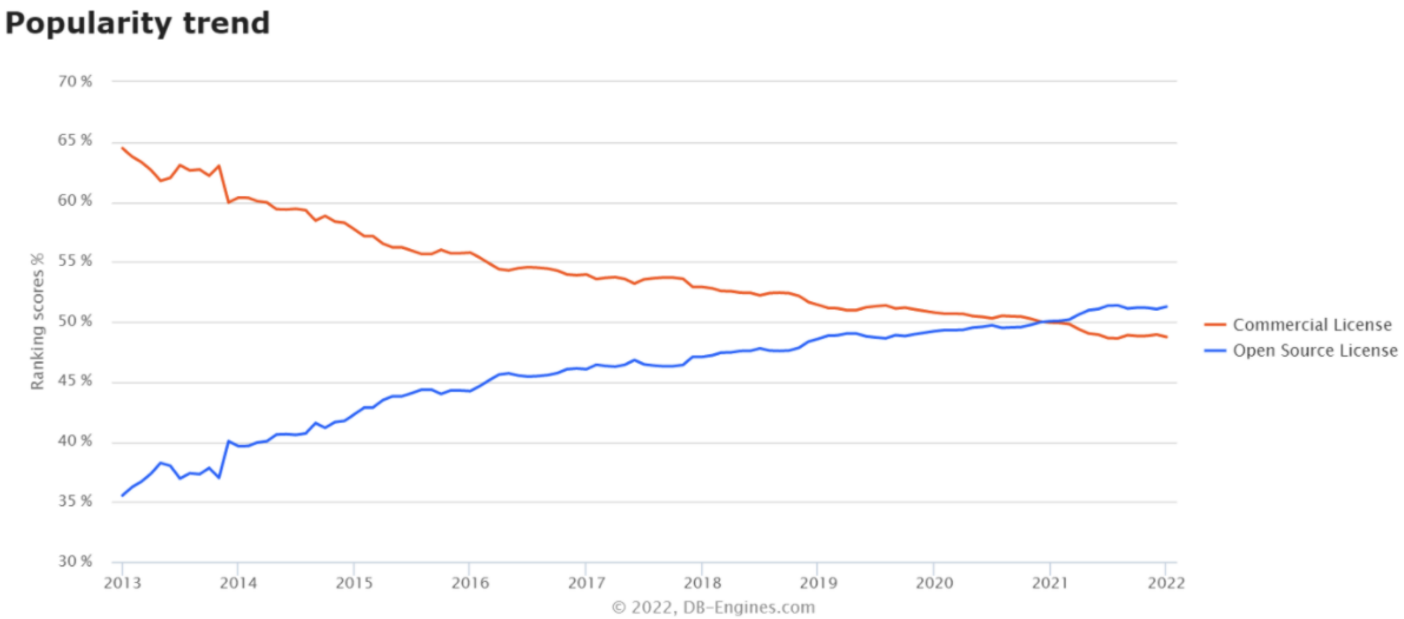
Lecture 03 - Select

Jan 25

Open Source Used More



Open source databases overtook closed source databases in popularity in early 2021 (DB-engines.com, link)

```
Select [ Projected Columns ]
From Table ...
    join to ...
Where ...
Group By Column List
Having [ where on grouped data ]
Ordr By [ Columns ]
```

Let’s use our table from last time

```
\c l02
create table vote_by_county (
    id             serial primary key,
    year           int default 2021,
    state          text default '--',        -- irritatingly all upper case.
    state_uc       text default '--',
    state_po       varchar(2) default '--',     -- Incorrectly Named Column!
    county_name    text default '--',        -- irritatingly all upper case.
    county_name_uc text default '--',
    county_fips    int default 0,
    office         text default 'unk',
    candidate      text default 'unk',
    candidate_uc   text default 'unk',
    party          text default 'unk',
    candidatevotes int default 0,
    totalvotes     int default 0,
    version        int,
    vote_mode      text
);
```

and do some selects with the projected columns.

```
select id, year
 from vote_by_county
;
```

we can rename a column

```
select id, year as "Year of Our Lord"
 from vote_by_county
;
```

we can pick different columns

File: 01.sql

```
select id, year, state, county_name as "county"
    from vote_by_county
;
```

Output:

```
  id   | year |        state         |        county
-------+------+----------------------+----------------------
    16 | 2000 | Alabama              | Bibb
  1992 | 2000 | Georgia              | Troup
  4932 | 2000 | Michigan             | Ingham
  5704 | 2000 | Mississippi          | Perry
 12884 | 2004 | Arkansas             | Pope
 ...
 72592 | 2020 | Wyoming              | Sheridan
 72593 | 2020 | Wyoming              | Sheridan
 72594 | 2020 | Wyoming              | Sublette
 72595 | 2020 | Wyoming              | Sublette
 72596 | 2020 | Wyoming              | Sublette
 72597 | 2020 | Wyoming              | Sublette
 72598 | 2020 | Wyoming              | Sweetwater
 72599 | 2020 | Wyoming              | Sweetwater
 72600 | 2020 | Wyoming              | Sweetwater
 72601 | 2020 | Wyoming              | Sweetwater
 72602 | 2020 | Wyoming              | Teton
 72603 | 2020 | Wyoming              | Teton
 72604 | 2020 | Wyoming              | Teton
 72605 | 2020 | Wyoming              | Teton
 72606 | 2020 | Wyoming              | Uinta
 72607 | 2020 | Wyoming              | Uinta
 72608 | 2020 | Wyoming              | Uinta
 72609 | 2020 | Wyoming              | Uinta
 72610 | 2020 | Wyoming              | Washakie
 72611 | 2020 | Wyoming              | Washakie
 72612 | 2020 | Wyoming              | Washakie
 72613 | 2020 | Wyoming              | Washakie
 72614 | 2020 | Wyoming              | Weston
 72615 | 2020 | Wyoming              | Weston
 72616 | 2020 | Wyoming              | Weston
 72617 | 2020 | Wyoming              | Weston
(72617 rows)
```

How about sorting the data

File: 02.sql

```
select id, year, state, county_name
    from vote_by_county
    order by county_name, state
;
```

Output:

```
  id   | year |        state         |      county_name
-------+------+----------------------+-----------------------
 28679 | 2008 | South Carolina       | Abbeville
 38030 | 2012 | South Carolina       | Abbeville
 38029 | 2012 | South Carolina       | Abbeville
 38028 | 2012 | South Carolina       | Abbeville
 28677 | 2008 | South Carolina       | Abbeville
  9148 | 2000 | South Carolina       | Abbeville
  9147 | 2000 | South Carolina       | Abbeville
  9146 | 2000 | South Carolina       | Abbeville
  9145 | 2000 | South Carolina       | Abbeville
 19328 | 2004 | South Carolina       | Abbeville
 19326 | 2004 | South Carolina       | Abbeville
 19327 | 2004 | South Carolina       | Abbeville
 28678 | 2008 | South Carolina       | Abbeville
 ...
 66634 | 2020 | Texas                | Zavala
 66635 | 2020 | Texas                | Zavala
 48759 | 2016 | Texas                | Zavala
 66633 | 2020 | Texas                | Zavala
 47714 | 2016 | South Dakota         | Ziebach
 69518 | 2020 | South Dakota         | Ziebach
 47713 | 2016 | South Dakota         | Ziebach
 47712 | 2016 | South Dakota         | Ziebach
 69516 | 2020 | South Dakota         | Ziebach
 69517 | 2020 | South Dakota         | Ziebach
 38362 | 2012 | South Dakota         | Ziebach
 38363 | 2012 | South Dakota         | Ziebach
 29012 | 2008 | South Dakota         | Ziebach
  9589 | 2000 | South Dakota         | Ziebach
 29011 | 2008 | South Dakota         | Ziebach
 29010 | 2008 | South Dakota         | Ziebach
 19661 | 2004 | South Dakota         | Ziebach
 19660 | 2004 | South Dakota         | Ziebach
```

```
  9590 | 2000 | South Dakota         | Ziebach
 19659 | 2004 | South Dakota         | Ziebach
  9591 | 2000 | South Dakota         | Ziebach
 38361 | 2012 | South Dakota         | Ziebach
  9592 | 2000 | South Dakota         | Ziebach
(72617 rows)
```

you can only sort by the columns that you have in the *projected columns*.

you can use the column position

File: 03.sql

```
select id, year, state, county_name
    from vote_by_county
    order by 4, 3
;
```

Output:

```
  id   | year |        state         |      county_name
-------+------+----------------------+-----------------------
 28679 | 2008 | South Carolina       | Abbeville
 38030 | 2012 | South Carolina       | Abbeville
 38029 | 2012 | South Carolina       | Abbeville
 38028 | 2012 | South Carolina       | Abbeville
 28677 | 2008 | South Carolina       | Abbeville
  9148 | 2000 | South Carolina       | Abbeville
  9147 | 2000 | South Carolina       | Abbeville
  9146 | 2000 | South Carolina       | Abbeville
  9145 | 2000 | South Carolina       | Abbeville
 19328 | 2004 | South Carolina       | Abbeville
 19326 | 2004 | South Carolina       | Abbeville
 19327 | 2004 | South Carolina       | Abbeville
 28678 | 2008 | South Carolina       | Abbeville
 47379 | 2016 | South Carolina       | Abbeville
 ...
 38362 | 2012 | South Dakota         | Ziebach
 38363 | 2012 | South Dakota         | Ziebach
 29012 | 2008 | South Dakota         | Ziebach
  9589 | 2000 | South Dakota         | Ziebach
 29011 | 2008 | South Dakota         | Ziebach
 29010 | 2008 | South Dakota         | Ziebach
 19661 | 2004 | South Dakota         | Ziebach
 19660 | 2004 | South Dakota         | Ziebach
  9590 | 2000 | South Dakota         | Ziebach
 19659 | 2004 | South Dakota         | Ziebach
  9591 | 2000 | South Dakota         | Ziebach
 38361 | 2012 | South Dakota         | Ziebach
  9592 | 2000 | South Dakota         | Ziebach
(72617 rows)
```

you can ascending or descending sort

File: 04.sql

```
select id, year, state, county_name
    from vote_by_county
    order by 4 desc, 3 asc
;
```

Output:

```
  id   | year |        state         |      county_name
-------+------+----------------------+-----------------------
  9591 | 2000 | South Dakota         | Ziebach
 19659 | 2004 | South Dakota         | Ziebach
 19660 | 2004 | South Dakota         | Ziebach
 19661 | 2004 | South Dakota         | Ziebach
 38361 | 2012 | South Dakota         | Ziebach
 38362 | 2012 | South Dakota         | Ziebach
 38363 | 2012 | South Dakota         | Ziebach
 29012 | 2008 | South Dakota         | Ziebach
 29011 | 2008 | South Dakota         | Ziebach
  9589 | 2000 | South Dakota         | Ziebach
 29010 | 2008 | South Dakota         | Ziebach
  9592 | 2000 | South Dakota         | Ziebach
  9590 | 2000 | South Dakota         | Ziebach
 47714 | 2016 | South Dakota         | Ziebach
 47713 | 2016 | South Dakota         | Ziebach
 47712 | 2016 | South Dakota         | Ziebach
 69518 | 2020 | South Dakota         | Ziebach
 69517 | 2020 | South Dakota         | Ziebach
 69516 | 2020 | South Dakota         | Ziebach
 48761 | 2016 | Texas                | Zavala
 66637 | 2020 | Texas                | Zavala
 66633 | 2020 | Texas                | Zavala
 ...
  9148 | 2000 | South Carolina       | Abbeville
 38030 | 2012 | South Carolina       | Abbeville
 38029 | 2012 | South Carolina       | Abbeville
 38028 | 2012 | South Carolina       | Abbeville
 28677 | 2008 | South Carolina       | Abbeville
 28678 | 2008 | South Carolina       | Abbeville
 28679 | 2008 | South Carolina       | Abbeville
 19326 | 2004 | South Carolina       | Abbeville
 19327 | 2004 | South Carolina       | Abbeville
(72617 rows)
```

You can apply functions and operators to the columns. In this case I will add 10 to the year and concatenate, `||` the state and county.

File: 05.sql

```
select id, year + 10 as "x", state||', '||county_name as "Location"
    from vote_by_county
    order by 3
;
```

Output:

```
   id   |  x   |                  Location
--------+------+-------------------------------------------
  31169 | 2022 | Alabama, Autauga
  31168 | 2022 | Alabama, Autauga
  31167 | 2022 | Alabama, Autauga
      1 | 2010 | Alabama, Autauga
  21818 | 2018 | Alabama, Autauga
  21817 | 2018 | Alabama, Autauga
  21816 | 2018 | Alabama, Autauga
  12465 | 2014 | Alabama, Autauga
  12466 | 2014 | Alabama, Autauga
  12467 | 2014 | Alabama, Autauga
      4 | 2010 | Alabama, Autauga
      3 | 2010 | Alabama, Autauga
      2 | 2010 | Alabama, Autauga
  40519 | 2026 | Alabama, Autauga
  50527 | 2030 | Alabama, Autauga
  40518 | 2026 | Alabama, Autauga
  50525 | 2030 | Alabama, Autauga
  ...
  49853 | 2026 | Wyoming, Uinta
  40502 | 2022 | Wyoming, Uinta
  40504 | 2022 | Wyoming, Washakie
  72610 | 2030 | Wyoming, Washakie
  72611 | 2030 | Wyoming, Washakie
  72612 | 2030 | Wyoming, Washakie
  72613 | 2030 | Wyoming, Washakie
  49856 | 2026 | Wyoming, Washakie
  40503 | 2022 | Wyoming, Washakie
  49855 | 2026 | Wyoming, Washakie
  49854 | 2026 | Wyoming, Washakie
  40505 | 2022 | Wyoming, Washakie
  12448 | 2010 | Wyoming, Washakie
  12447 | 2010 | Wyoming, Washakie
  12445 | 2010 | Wyoming, Washakie
  31152 | 2018 | Wyoming, Washakie
  31153 | 2018 | Wyoming, Washakie
  21803 | 2014 | Wyoming, Washakie
  21802 | 2014 | Wyoming, Washakie
  31154 | 2018 | Wyoming, Washakie
  21801 | 2014 | Wyoming, Washakie
  12446 | 2010 | Wyoming, Washakie
  12452 | 2010 | Wyoming, Weston
  12449 | 2010 | Wyoming, Weston
  31156 | 2018 | Wyoming, Weston
  31155 | 2018 | Wyoming, Weston
  12451 | 2010 | Wyoming, Weston
  21806 | 2014 | Wyoming, Weston
  21805 | 2014 | Wyoming, Weston
  21804 | 2014 | Wyoming, Weston
  12450 | 2010 | Wyoming, Weston
  31157 | 2018 | Wyoming, Weston
  49859 | 2026 | Wyoming, Weston
  72616 | 2030 | Wyoming, Weston
  40507 | 2022 | Wyoming, Weston
  49858 | 2026 | Wyoming, Weston
  49857 | 2026 | Wyoming, Weston
  72615 | 2030 | Wyoming, Weston
```

```
 40506 | 2022 | Wyoming, Weston
 72617 | 2030 | Wyoming, Weston
 72614 | 2030 | Wyoming, Weston
 40508 | 2022 | Wyoming, Weston
(72617 rows)
```

Single quotes denote string constants. Double quotes denote things like tables and column names. If you want an upper-lower case or a table name or column name with blanks then you have to quote it with double quotes.

Lot’s of stuff will fail if you put blanks in your table names. This is worse than putting blanks in your file names. Python will crash with blanks in file names.

We will be using more than one table in queries. To do this we need to tell SQL the table. That is done with a table-alias. In this case *t1*.

File: 06.sql

```
select t1.id, t1.year + 10 as "x", t1.state||', '||t1.county_name as "Location"
    from vote_by_county  as t1
    order by 3 asc
;
```

There are lots of builtin functions that you can use and all sorts of arithmetic operators in PostgreSQL that can be applied to the projected columns. You can also write your own functions and pass data from the projected columns to the function and get back results.

Languages for functions include PG/SQL - the built in PostgreSQL language and others like JavaScript, Lua, C, C++, Go etc. I use PG/SQL and C for processing. JavaScript is 5 to 10x slower than PG/SQL. C is 10x faster than PG/SQL. These are rough numbers. I am a big fan of Go but I haven’t used it for stored-procedure/functions yet in PostgreSQL.

Using a C function requires re-loading and re-starting the database - so it is hard.

Get the list of all of the candidates in the data.

```
select distinct candidate from vote_by_county ;
```

Find out what counties where voted in:

```

-- Find the set of counties that voted for "Sleepy Joe" in 2020

select t1.state, t1.county_name
    from vote_by_county as t1
    where t1.year = 2020
        and t1.candidate = 'Joseph R Biden Jr'
    order by state, county_name
;

select t1.state, t1.county_name
```

```
    from vote_by_county as t1
    where t1.year = 2020
        and t1.candidate = 'Donald J Trump'
    order by state, county_name
;

```

The question is who won? To find that we have to find the candidate that had the most votes in each county. We need to use the “max” in a county.

```

-- Find the set of counties that voted for "Sleepy Joe" in 2020

select t1.state, t1.county_name
from vote_by_county as t1
where t1.year = 2020
    and t1.candidate = 'Joseph R Biden Jr'
    and t1.candidatevotes = (
        select max(t2.candidatevotes) as max_votes
        from vote_by_county as t2
        where t2.state = t1.state
          and t2.county_name = t1.county_name
    )
    order by state, county_name
    ;


```

Operators

https://www.postgresql.org/docs/9.0/functions.html

There are lots!

Operator	Description	Example	Result
+	addition	2 + 3	5
-	subtraction	2 - 3	-1
*	multiplication	2 * 3	6
/	division (integer division truncates the result)	4 / 2	2
%	modulo (remainder)	5 % 4	1
^	exponentiation	2.0 ^ 3.0	8
|/	square root	|/ 25.0	5
||/	cube root	||/ 27.0	3
!	factorial	5 !	120
!!	factorial (prefix operator)	!! 5	120

Operator	Description	Example	Result
@	absolute value	@ -5.0	5
&	bitwise AND	91 & 15	11
|	bitwise OR	32 | 3	35
#	bitwise XOR	17 # 5	20
~	bitwise NOT	~1	-2
<<	bitwise shift left	1 << 4	16
>>	bitwise shift right	8 >> 2	2

and string operations

Function	Return Type	Description	Example	Result
string || string	text	String concatenation	'Post' || 'greSQL'	PostgreSQL
string || non-string or non-string || string	text	String concatenation with one non-string input	'Value: ' || 42	Value: 42
bit_length(string)	int	Number of bits in string	bit_length('jose')	32
char_length(string) or character_length(string)	int	Number of characters in string	char_length('jose')	4
lower(string)	text	Convert string to lower case	lower('TOM')	tom
octet_length(string)	int	Number of bytes in string	octet_length('jose')	4
overlay(string placing string from int [for int])	text	Replace substring	overlay('Txxxxas' placing 'hom' from 2 for 4)	Thomas
position(substring in string)	int	Location of specified substring	position('om' in 'Thomas')	3
substring(string [from int] [for int])	text	Extract substring	substring('Thomas' from 2 for 3)	hom
substring(string from pattern)	text	Extract substring matching POSIX regular expression. See Section 9.7 for more information on pattern matching.	substring('Thomas' from '...$')	mas
substring(string from pattern for escape)	text	Extract substring matching SQL regular expression. See Section 9.7 for more information on pattern matching.	substring('Thomas' from '%#"o_a#"_' for '#')	oma
trim([leading | trailing | both] [characters] from string)	text	Remove the longest string containing only the characters (a space by default) from the start/end/both ends of the string	trim(both 'x' from 'xTomxx')	Tom

Function	Return Type	Description	Example	Result
`upper(string)`	`text`	Convert string to upper case	`upper('tom')`	`TOM`

Base Functions

Function	Return Type	Description	Example	Result
`abs(x)`	(same as input)	absolute value	`abs(-17.4)`	`17.4`
`cbrt(dp)`	dp	cube root	`cbrt(27.0)`	`3`
`ceil(dp or numeric)`	(same as input)	smallest integer not less than argument	`ceil(-42.8)`	`-42`
`ceiling(dp or numeric)`	(same as input)	smallest integer not less than argument (alias for `ceil`)	`ceiling(-95.3)`	`-95`
`degrees(dp)`	dp	radians to degrees	`degrees(0.5)`	`28.6478897565412`
`div(y numeric, x numeric)`	`numeric`	integer quotient of `y`/`x`	`div(9,4)`	`2`
`exp(dp or numeric)`	(same as input)	exponential	`exp(1.0)`	`2.71828182845905`
`floor(dp or numeric)`	(same as input)	largest integer not greater than argument	`floor(-42.8)`	`-43`
`ln(dp or numeric)`	(same as input)	natural logarithm	`ln(2.0)`	`0.693147180559945`
`log(dp or numeric)`	(same as input)	base 10 logarithm	`log(100.0)`	`2`
`log(b numeric, x numeric)`	`numeric`	logarithm to base `b`	`log(2.0, 64.0)`	`6.0000000000`
`mod(y, x)`	(same as argument types)	remainder of `y`/`x`	`mod(9,4)`	`1`
`pi()`	dp	"π" constant	`pi()`	`3.14159265358979`
`power(a dp, b dp)`	dp	a raised to the power of `b`	`power(9.0, 3.0)`	`729`
`power(a numeric, b numeric)`	`numeric`	a raised to the power of `b`	`power(9.0, 3.0)`	`729`
`radians(dp)`	dp	degrees to radians	`radians(45.0)`	`0.785398163397448`
`round(dp or numeric)`	(same as input)	round to nearest integer	`round(42.4)`	`42`
`round(v numeric, s int)`	`numeric`	round to s decimal places	`round(42.4382, 2)`	`42.44`
`sign(dp or numeric)`	(same as input)	sign of the argument (-1, 0, +1)	`sign(-8.4)`	`-1`
`sqrt(dp or numeric)`	(same as input)	square root	`sqrt(2.0)`	`1.4142135623731`

Function	Return Type	Description	Example	Result
`trunc(dp or numeric)`	(same as input)	truncate toward zero	`trunc(42.8)`	`42`
`trunc(v numeric, s int)`	`numeric`	truncate to `s` decimal places	`trunc(42.4382, 2)`	`42.43`
`width_bucket(op numeric, b1 numeric, b2 numeric, count int)`	`int`	return the bucket to which `operand` would be assigned in an equidepth histogram with `count` buckets, in the range `b1` to `b2`	`width_bucket(5.35, 0.024, 10.06, 5)`	`3`
`width_bucket(op dp, b1 dp, b2 dp, count int)`	`int`	return the bucket to which `operand` would be assigned in an equidepth histogram with `count` buckets, in the range `b1` to `b2`	`width_bucket(5.35, 0.024, 10.06, 5)`	`3`

How About the square root operator!

File: 07.sql

```
select |/25.0;
```

and a factorial operator!

```
select 5!;
```

and

```
select !! 5;
```

That is *FUN* ! Not 1 but 2 factorial operators.