

# Lecture 04 - More Select

---

Thu Jan 27 2022

---

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

Ok.... We have looked at the projected columns and order by.

Let's use our table from last time

```
1: \c l02
2: create table vote_by_county (
3:     id                serial primary key,
4:     year               int default 2021,
5:     state              text default '--',          -- irritatingly all upper case.
6:     state_uc           text default '--',
7:     state_po           varchar(2) default '--',    -- Incorrectly Named Column!
8:     county_name        text default '--',          -- irritatingly all upper case.
9:     county_name_uc     text default '--',
10:    county_fips         int default 0,
11:    office              text default 'unk',
12:    candidate           text default 'unk',
13:    candidate_uc        text default 'unk',
14:    party               text default 'unk',
15:    candidatevotes      int default 0,
16:    totalvotes          int default 0,
17:    version             int,
18:    vote_mode           text
19: );
```

Let's just insert a few rows to see how insert works:

```
1: -- \c l02
2:
3: insert into vote_by_county ( year, state, county_name, version ) values
4:     ( 2022, 'Wyoming', 'Albeny', 1 );
5: insert into vote_by_county ( year, state, county_name, version ) values
6:     ( 2022, 'Wyoming', 'Big Horn', 2 );
7: insert into vote_by_county ( year, state, county_name, version ) values
8:     ( 2022, 'Wyoming', 'Carbon', 8 );
```

and look into the "where"

```
select id, year
      from vote_by_county
     where county_name = 'Carbon'
;
```

or a list

```
select id, year
      from vote_by_county
     where county_name in ( 'Carbon', 'Albeny' )
;
```

we or and and .

```
select id, year, state, county
      from vote_by_county
     where county_name = 'Carbon'
        or county_name = 'Albeny'
;
```

Comparison with operators

```
select id, year, state, county
      from vote_by_county
     where version < 4
;
```

This is where other tools (ORMs, MongoDB etc) fail : they only allow you to pick stuff that is by example, as in equal to.

```
select id, year, state, county
      from vote_by_county
     where totalvotes != 0
;
```

To really understand this we need more than 3 rows of data. We will have a lecture on 'copy'/'to' and 'copy'/'from' but let's load some data from lecture 2 and start really using the where clause.

```
delete from vote_by_county ;

\COPY vote_by_county ( year, state_uc, state_po, county_name_uc, county_fips, office,
candidate_uc, party, candidatevotes, totalvotes, version, vote_mode ) FROM
'countypres_2000-2020.csv' DELIMITER ',' NULL AS 'NA' CSV HEADER;
```

Note that \COPY is not the same as COPY - and \COPY really has to be on a single line.

one of the functions that we can use in the projected columns is count(1) or count(\*) . They have different performance characteristics.

```
select count(1) as "number of rows" from vote_by_county ;
```

## Update

---

some quick fixes - we will cover 'Update' a little later too..

```
update vote_by_county
    set state = initcap ( state_uc )
    ;
update vote_by_county
    set county_name = initcap ( county_name_uc )
    ;
update vote_by_county
    set candidate = initcap ( candidate_uc )
    ;
```

## And now back to select and operators

---

And now let's apply this for all of a single candidate. This will give us all the counties that the candidate won.

```
1:
2: -- Find the set of counties that voted for "Sleepy Joe" in 2020
3:
4: select t1.state, t1.county_name
5: from vote_by_county as t1
6: where t1.year = 2020
7:     and t1.candidate = 'Joseph R Biden Jr'
8:     and t1.candidatevotes = (
9:         select max(t2.candidatevotes) as max_votes
10:        from vote_by_county as t2
11:        where t2.state = t1.state
12:        and t2.county_name = t1.county_name
13:    )
14: order by state, county_name
15: ;
```

And the former president

```
1: select t1.state, t1.county_name
2: from vote_by_county as t1
3: where t1.year = 2020
4:     and t1.candidate = 'Donald J Trump'
5:     and t1.candidatevotes = (
6:         select max(t2.candidatevotes) as max_votes
7:        from vote_by_county as t2
8:        where t2.state = t1.state
9:        and t2.county_name = t1.county_name
10:    )
11: order by state, county_name
12: ;
```

## 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
@	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

## ### string operations

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

## Base Functions

Function	Return Type	Description	Example	Result
<code>abs(x)</code>	(same as input)	absolute value	<code>abs(-17.4)</code>	17.4
<code>cbrt(dp)</code>	dp	cube root	<code>cbrt(27.0)</code>	3
<code>ceil(dp or numeric)</code>	(same as input)	smallest integer not less than argument	<code>ceil(-42.8)</code>	-42
<code>ceiling(dp or numeric)</code>	(same as input)	smallest integer not less than argument (alias for <code>ceil</code> )	<code>ceiling(-95.3)</code>	-95
<code>degrees(dp)</code>	dp	radians to degrees	<code>degrees(0.5)</code>	28.6478897565412
<code>div(y numeric, x numeric)</code>	numeric	integer quotient of y/x	<code>div(9,4)</code>	2
<code>exp(dp or numeric)</code>	(same as input)	exponential	<code>exp(1.0)</code>	2.71828182845905
<code>floor(dp or numeric)</code>	(same as input)	largest integer not greater than argument	<code>floor(-42.8)</code>	-43
<code>ln(dp or numeric)</code>	(same as input)	natural logarithm	<code>ln(2.0)</code>	0.693147180559945
<code>log(dp or numeric)</code>	(same as input)	base 10 logarithm	<code>log(100.0)</code>	2
<code>log(b numeric, x numeric)</code>	numeric	logarithm to base b	<code>log(2.0, 64.0)</code>	6.0000000000
<code>mod(y, x)</code>	(same as argument types)	remainder of y/x	<code>mod(9,4)</code>	1
<code>pi()</code>	dp	" $\pi$ " constant	<code>pi()</code>	3.14159265358979
<code>power(a dp, b dp)</code>	dp	a raised to the power of b	<code>power(9.0, 3.0)</code>	729
<code>power(a numeric, b numeric)</code>	numeric	a raised to the power of b	<code>power(9.0, 3.0)</code>	729
<code>radians(dp)</code>	dp	degrees to radians	<code>radians(45.0)</code>	0.785398163397448
<code>round(dp or numeric)</code>	(same as input)	round to nearest integer	<code>round(42.4)</code>	42
<code>round(v numeric, s int)</code>	numeric	round to s decimal places	<code>round(42.4382, 2)</code>	42.44
<code>sign(dp or numeric)</code>	(same as input)	sign of the argument (-1, 0, +1)	<code>sign(-8.4)</code>	-1
<code>sqrt(dp or numeric)</code>	(same as input)	square root	<code>sqrt(2.0)</code>	1.4142135623731
<code>trunc(dp or numeric)</code>	(same as input)	truncate toward zero	<code>trunc(42.8)</code>	42
<code>trunc(v numeric, s int)</code>	numeric	truncate to s decimal places	<code>trunc(42.4382, 2)</code>	42.43

Function	Return Type	Description	Example	Result
<code>width_bucket( numeric, b1 numeric, b2 numeric, count int)</code>	int	return the bucket to which operand would be assigned in an equidepth histogram with count buckets, in the range b1 to b2	<code>width_bucket(5.35, 0.024, 10.06, 5)</code>	3
<code>width_bucket(op dp, b1 dp, b2 dp, count int)</code>	int	return the bucket to which operand would be assigned in an equidepth histogram with count buckets, in the range b1 to b2	<code>width_bucket(5.35, 0.024, 10.06, 5)</code>	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.