Lecture 04 - More Select

Jan 27

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

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

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
);
```

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

```
-- \c l02

insert into vote_by_county  ( year, state, county_name, version ) values
    ( 2022, 'Wyoming', 'Albeny',   1 );
insert into vote_by_county  ( year, state, county_name, version ) values
    ( 2022, 'Wyoming', 'Big Horn', 2 );
insert into vote_by_county  ( year, state, county_name, version ) values
    ( 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.

And the former president

```
select t1.state, t1.county_name
from vote_by_county as t1
where t1.year = 2020
        and t1.candidate = 'Donald J Trump'
        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
@	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
`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`
`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`
`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`

Function	Return Type	Description	Example	Result
`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

and a factorial operator!

```
select 5!;
```

and

```
select !! 5;
```

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