New SQL for binning

Method 1: Group by extracted time component

This SQL statement is computing average of the bins which created by grouping the extracted time component

```
geostream=# select extract(year from start_time) as yyyy, avg(cast( data ->> 'temperature' as DOUBLE PRECISION)) from datapoints where stream_id = 18133 group by 1;
 yyyy |       avg
------+------------------
 2003 | 1.29503442622951
(1 row)
```

Explanation of the SQL:

"where" statement contains the filtering, it should include the stream_id (sensor_id), could include start time, end time, source .....

"group by" is to do the grouping, "by 1" meaning group by the first selection, i.e. extract(year from start_time) in this example. it can also by "by 1,2"....

"cast(data ->> 'temperature' as DOUBLE PRECISION)" is to find the 'temperature' in data and convert it to double format. Using the following SQL can make it more responsive (just running once):

```
create or replace function cast_to_double(text) returns DOUBLE PRECISION as $$


begin
    -- Note the double casting to avoid infinite recursion.
    return cast($1::varchar as DOUBLE PRECISION);
exception
    when invalid_text_representation then
        return 0.0;
end;
$$ language plpgsql immutable;
create cast (text as DOUBLE PRECISION) with function cast_to_double(text);
```

"avg" is to get the average, it usually used alone with "group by", you can have sum, count.....

Overview

by returning the average of the datapoints, short the time for streaming.

then we just need to convert each SQL result to a json.

Limitation

- Monthly, daily binning will not work. For example, it will group all "December" regardless of year.
- Customized binning, such as water years, can not be used.

Method 2: Group by pre-generated bins with join

This SQL statement pre-generate bins using "generate_series()" and tstzrange type; then it joins with datapoints table and groups by bins

```
with bin as (
        select tstzrange(s, s+'1 year'::interval) as r
        from generate_series('2002-01-01 00:00:00-05'::timestamp, '2017-12-31 23:59:59-05'::timestamp, '1
year') as s
)
select
        bin.r,
        avg(cast( data->> 'pH' as DOUBLE PRECISION))
from datapoints
right join bin on datapoints.start_time <@ bin.r
where datapoints.stream_id = 1584
group by 1
order by 1;
```

Method 3: Group by pre-generated bins with filter for aggregate function

This SQL statement pre-generate bins using "generate_series()" and tstzrange type; then it uses filter with avg function instead of join

```
with bin as (
        select tstzrange(s, s+'1 year'::interval) as r
        from generate_series('2002-01-01 00:00:00-05'::timestamp, '2017-12-31 23:59:59-05'::timestamp, '1
year') as s
)
select
        bin.r,
        avg(cast( data->> 'pH' as DOUBLE PRECISION)) filter(where datapoints.start_time <@ bin.r)
from datapoints, bin
where datapoints.stream_id = 1584
group by 1
order by 1;
```

Method 4: Using aggregate function with "over" and "window" (not working yet)

Jong Lee Looked into this option; but couldn't find a way to do it. Jong Lee may not understand the functionality.

Method 5: Using procedure function (PL/pgsql or PL/python)

TODO...

Performance

Tested with GLGT production database. Used the stream_id 1584 which has 987,384 datapoints. Used "explain analyze"

	Method 1	Method 2	Method 3
Planning time	0.269 ms	0.259 ms	0.145 ms
Execution time	3069.059 ms	6029.105 ms	12008.801 ms

Caching design

If we store the count and sum of the values in addition to average, it becomes easy to update the bin with a new datapoint.

bins_year

sensor_id	year	field	count	sum	average	start_time	end_time	updated
12345	2003	temperature	120	8400.0	60.0			
12345	2003	pH	120	240.0	2.0			

- are start/end times for bins actually useful for anything? there could be holes in between endpoints
- store completeness by sensor /stream?

bins_month

sensor_id	month	year	field	count	sum	average	start_time	end_time	updated
12345	6	2003	temperature	10	600.0	60.0			
12345	6	2003	pH	10	20.0	2.0			

bins_day

sensor_id	day	month	year	field	count	sum	average	start_time	end_time	updated
12345	13	6	2003	temperature	1	60.0	60.0			
12345	13	6	2003	pH	1	2.0	2.0			

bins_hour

sensor_id	hour	day	month	year	field	count	sum	average	start_time	end_time	updated
12345	19	13	6	2003	temperature	1	60.0	60.0			
12345	19	13	6	2003	pH	1	2.0	2.0			

bins_special

sensor_id	label	field	count	sum	average	start_time	end_time	updated
12345	spring	temperature	3	180.0	60.0	01/01/2003	03/31/2003	
12345	spring	pH	3	6.0	2.0	01/01/2003	03/31/2003	
12345	spring	pH	1	2.0	2.0	01/01/2003	01/31/2003	

for bins_special, do we actually need count/sum/average here, or does it simply need a start/end time and an aggregation level (year/month/etc) that defines the custom aggregation unit and use the bins_year, bins_month to populate?

- for spring, we get monthly averages only within start/end time
- for spring, we get yearly average only including months within start/end

bins_special (alt option)

label	start_time	end_time	updated
spring	Jan 1	Mar 31	
spring	Jan 1	Mar 31	

- if i want special bin by year, only consider points between start and end time.
    - if start/end time includes entire year, use bins_year
    - if < 1 year time span, aggregate month + day bins until you cover entire time span
- if i want by months, include each month between start/end time
    - for complete months, use bins_month

- for partial months, aggregate day bins until you cover entire time span

Other possible tables:

bins_season - do we need to cache this, or calculate from monthly bins? latter option suggested above.

bins_total - do we need to cache this, or is it fast enough to calculate from yearly bins?

I don't think we want cache table for water_year, for example, because that is specific to GLM/GLTG and not generic for clowder. We could use the month caches to quickly calculate that on the fly.

When do we update cache tables?

- cron job (hourly? 5 minutes?)
- whenever new datapoint is added (at most 1 bin per table would need to be created or updated) - upsert

Trends region

```
seagrant-dev=# select * from regions;
 id |   title    |                                                boundary
                                        |              center_coordinate
----+------------+------------------------------------------------------------------------------------------------
 hu | Lake Huron | 0103000020E610000001000000040000000000000000482A55C0AF047D0F6C0747400000000000500455C04BBE29C8EB3B47400000000000702055C0E92224C7
98CE4640000000000000482A55C0AF047D0F6C074740 | 0101000020E61000000000000000008054C00000000000804640
(1 row)

seagrant-dev=# select * from region_trends;
 region_id | season |      parameter      |    lastaverage    |  tenyearsaverage  |   totalaverage
-----------+--------+---------------------+-------------------+-------------------+-------------------
 hu        | spring | chlorophyll-a-glenda | 0.584240707964602 | 0.525581579414375 | 0.524933337142857
(1 row)
```