
Pivotal
A NEW PLATFORM FOR A NEW ERA

User Defined Types
User Defined Aggregates

Pivotal® Greenplum
Database

Agenda

- User Defined Types
  - Why use them
  - How to create them
- User Defined Aggregates
  - Why use them
  - How to create them
  - How to use them

User Defined Types

- Simplifies the access to a related set of properties
- Builds an abstraction layer that's easier to maintain
- Allows access to the individual members of the UDT
- Similar to a C language struct
- Example:

```
create type address_type as (
        street text,
        city text,
        state char(2),
        zipcode char(5);
```

Pivotal

User Defined Types -- Usage

```
create table label (
   name text,
   address address_type,
   order_no integer) distributed randomly;
insert into label values
   ('Fred Derf', ('POBox 123','Woof', 'NY', '12345'), 9991);
insert into label values
    ('Big Daddy', ('99 Main St.','Yourtown', 'WY', '88888'), 4388);

select (address).state, (address).zipcode, order_no from label;
  state | zipcode | order_no
  -------+---------+----------
  NY    | 12345   |     9991
  WY    | 88888   |     4388
```

Pivotal

User Defined Aggregates

- An extension of the built in aggregate functions
  - min, max, count, sum, avg, count(*), etc.
- Unlike window functions which return a value for each row, aggregates return one value of a set of rows.
- Useful for non-standard operations important in an application.
- Provide maintainability and code hiding from users.
- Take advantage of parallelism in Greenplum

User Defined Aggregate -- creation

UDAs are defined in terms of

- State type – what data type is being aggregated
- State transition functions – how is it being aggregated
- Initial condition – what value do we start with
- Final function – how to terminate the process (optional)

Pivotal

User Defined Aggregate – Simple Example

```
create or replace function demo.int_add_state(integer,  integer) returns integer as
    'select $1 + $2;'
 language sql immutable;

create aggregate demo.mysum (integer)
 ( stype = integer,
   sfunc = demo.int_add_state,
   initcond = 0
);
select * from demo.sales where prod = ‘ball’
 sales_date |  prod   | qty
------------+---------+-----
 2016-01-01 |  'ball' |  11
 2016-01-02 |  'ball' |  22
 2016-01-03 |  'ball' |  33
select demo.mysum(qty) from demo.sales;
 mysum
-------
66
```

Pivotal

User Defined Aggregates – Example

United Widget has a production process that makes widgets in lots and runs tests on them during the process. In addition to the problem above, their analysts need to compute the geometric mean of one of the tests.

If there are N test scores, the geometric mean is the Nth root of the product of the numbers. The computationally common way to compute this is to take the arithmetic mean of the natural log of the numbers and then compute the exponent of that.

You could write that in SQL as:

```
select exp(avg(ln(sample)))
  from sample_table
```

but what fun is that? Let's write it as an aggregate.

Pivotal

Sample data

```
gpadmin=# select * from tests;
    t1      |   t2   |     t3       |     t4
------------+--------+--------------+-------------
  -11930.5 |  -3758 | 12508.37012 | 197.7077942
   36831.5 |  88012 | 95407.92188 | 67.19883728
  -85073.5 | -95528 | 127918.2969 |    228.4944
  -60692.5 |  88012 | 106909.7031 | 124.4182968
   12450.5 | 118602 | 119253.7031 | 83.89148712
   85593.5 |  26832 | 89700.63281 |  17.3812809
 -133835.5 |  -3758 | 133888.2969 | 181.8540955
  -60692.5 |  57422 | 83551.57031 | 136.3979034
  -36311.5 | -34348 | 49983.10156 |  223.596405
   12450.5 | -64938 | 66120.78906 | 280.9625854
(10 rows)
```

Geometric mean state function

```
create or replace function geom_mean_state(double precision[2],
double precision)
returns double precision[] as
$$
select case when $2 is NULL or $2 <= 0.0  then $1
    else array[coalesce($1[1],0)+ ln($2), $1[2] + 1.0] end;
$$
language sql immutable;
```

- Note: the geometric mean calculation only works for positive values.
- The array contains the running sum of the logs in the 0'th element and the running count of the number of items in the 1'st element.
- $1 is the running total so far and $2 is the new value.

Pivotal

Geometric mean final function

```
create or replace function geom_mean_final(float8[2])
returns float8 as
$$
select case when $1[2] > 0 then exp($1[1]/$1[2]) else 0 END;
$$
language sql immutable;
```

- The final function then takes raises uses the exp function on the sum of the logs divided by the number of elements scanned in the state function.

Pivotal

Geometric mean aggregate definition

```
create aggregate geom_mean(float8) (
  SFUNC = geom_mean_state,
  STYPE = float8[],
  FINALFUNC = geom_mean_final,
  INITCOND = '{0.0,0.0}'
);
```

- The UDA (user defined aggregate) initializes the array to zeros, defines the data type of the function, and specifies the state function and final function.

Pivotal

Run geom_mean aggregate and standard computation
They should give identical results

```
select geom_mean(t4) from tests;
    geom_mean
------------------
 123.982078311074
(1 row)

Time: 29.433 ms
select exp(avg(ln(t4)))
  from tests;
        exp
------------------
 123.982078311074
(1 row)
```

Original Problem

Love is fickle. The manager at United Widgets once enamored with harmonic means, now has read about moving averages and they have captured his heart.

A moving average of a time series is the average of some the values of time periods before the present

The n days moving average is the arithmetic average of the value of a variable during the last n days.

With:

P_t the value at the date t.

P_{t-1} the value at the date t-1.

MA_{tn} the n days moving average at the date t.

The n days moving average can be computed as follows: $$MA_n = \frac{\sum_{i=0}^{n-1} P_{t-i}}{n}$$

Sample Data -- defect rate daily

```
select dt, dfct_rate from defects order by 1 limit 15;
         dt          |     dfct_rate
---------------------+--------------------
 2012-01-01 12:00:00 |             3.164
 2012-01-02 12:00:00 |             3.506
 2012-01-03 12:00:00 |             0.518
 2012-01-04 12:00:00 |             0.734
 2012-01-05 12:00:00 |             2.354
 2012-01-06 12:00:00 |             1.094
 2012-01-07 12:00:00 |              2.21
 2012-01-08 12:00:00 |             1.994
 2012-01-09 12:00:00 |              2.75
 2012-01-10 12:00:00 |             3.452
 2012-01-11 12:00:00 |             0.914
 2012-01-12 12:00:00 |             2.156
 2012-01-13 12:00:00 |             2.336
 2012-01-14 12:00:00 |              0.05
 2012-01-15 12:00:00 | 0.0399999999999999
```

Moving average of defects over 4 days

```
select
  dt, dfct_rate,
  avg(dfct_rate)
    over (order by dt asc
          rows between 3 preceding and current row) as mvg_avg
from defects order by 1 asc;
```

dt	dfct_rate	mvg_avg
2012-01-01 12:00:00	3.164	3.164
2012-01-02 12:00:00	3.506	3.335
2012-01-03 12:00:00	0.518	2.396
2012-01-04 12:00:00	0.734	1.9805
2012-01-05 12:00:00	2.354	1.778
2012-01-06 12:00:00	1.094	1.175
2012-01-07 12:00:00	2.21	1.598
2012-01-08 12:00:00	1.994	1.913
2012-01-09 12:00:00	2.75	2.012
2012-01-10 12:00:00	3.452	2.6015
2012-01-11 12:00:00	0.914	2.2775
2012-01-12 12:00:00	2.156	2.318
2012-01-13 12:00:00	2.336	2.2145
2012-01-14 12:00:00	0.05	1.364
2012-01-15 12:00:00	0.0399999999999999	1.1455

Pivotal

UDA with a Window Function

"But wait", says the manager at United Widgets. "I didn't want the arithmetic mean, I wanted the geometric mean. This is a simple change, just change the avg function above to the geom_mean user defined aggregate we already developed and get back to me pronto. This is so simple, I wrote it for you."

```
select
  dt, dfct_rate,
   geom_mean(dfct_rate)
    over (order by dt asc
          rows between 3 preceding and current row) as
mvg_geom_mean from defects order by 1 asc;
```

Pivotal

That Didn't Work!

```
select
  dt,
  dfct_rate,
   geom_mean(dfct_rate)
    over (order by dt asc
          rows between 3 preceding and current row) as
mvg_geom_mean
from defects
order by 1 asc;
```

ERROR: aggregate functions with no prelimfn or invprelimfn are not yet supported as window functions

Pivotal

The Postgres doc set is of no help. What's going on?

- No version of the Postgres documentation mentions anything about prelimfn or invprelimfn.
- This is a Pivotal Greenplum Database feature not in any version of Postgres. See the CREATE AGGREGATE page in the GPDB43RefGuide available at:

  http://gpdb.docs.pivotal.io/4351/pdf/GPDB43RefGuide.pdf

Pivotal

How the GPDB implements aggregates

The GPDB aggregate definition allows user to specify a PREFUNC. The individual segments run the state change function to do partial aggregation. The prefunc is run on the master to aggregate the results obtained by from the final results of the state change functions on the segments.

This enables the query optimizer to generate two phase aggregation plans which allows individual segments to perform partial aggregation prior to redistributing data to other segments or gathering data to the master node, this in turn can lead to improved parallelism and reduced data movement for many types of query plans.

The final function is then executed on the master.

Pivotal

Geometric mean with a prelimfunc

The state transition function and final function are the same:

```
create or replace function geom_mean_state(float8[2], float8)
returns float8[] as
$$
select case when $2 is NULL or $2 <= 0.0  then $1
    else array[coalesce($1[1],0)+ ln($2), $1[2] + 1.0] end;
$$
language sql immutable;

create or replace function geom_mean_final(float8[2])
returns float8 as
$$
select case when $1[2] > 0 then exp($1[1]/$1[2]) else 0 END;
$$
language sql immutable;
```

Pivotal

Geometric mean with a prelimfunc (cont.)

The prefunc simply aggregates the terms from the state change function on the segments

```
create or replace function geom_mean_prefunc(float8[2],float8[2])
returns float8[2] as
$$
select array[$1[1]+$2[1], $1[2]+$2[2]]
$$
language sql immutable;
```

And the aggregate definition adds the name of the prefunc

```
create aggregate geom_mean(float8) (
  SFUNC = geom_mean_state,
  STYPE = float8[],
  FINALFUNC = geom_mean_final,
  PREFUNC = geom_mean_prefunc,
  INITCOND = '{0.0,0.0}'
);
```

Pivotal

And the result is …

```
select
  dt,
  dfct_rate,
   geom_mean(dfct_rate)
    over (order by dt asc
          rows between 3 preceding and current row) as mvg_geom_mean
from defects
order by 1 asc;
```

dt	dfct_rate	mvg_geom_mean
2012-01-01 12:00:00	3.164	3.164
2012-01-02 12:00:00	3.506	3.33061315676258
2012-01-03 12:00:00	0.518	1.79112563138236
2012-01-04 12:00:00	0.734	1.43307378489496
2012-01-05 12:00:00	2.354	1.33094787593113
2012-01-06 12:00:00	1.094	0.994746454828728
2012-01-07 12:00:00	2.21	1.42964440354648
2012-01-08 12:00:00	1.994	1.83541933342358
2012-01-09 12:00:00	2.75	1.90816910907057
2012-01-10 12:00:00	3.452	2.54319831481974
2012-01-11 12:00:00	0.914	2.03947651009685
2012-01-12 12:00:00	2.156	2.07969481128572
2012-01-13 12:00:00	2.336	1.99657047451008
2012-01-14 12:00:00	0.05	0.692642998806025
2012-01-15 12:00:00	0.0399999999999999	0.316801987590269

Pivotal

References

- All the Pivotal Greenplum Database documentation is available online
  - Reference Guide
  - Admin Guide
  - Client Tools
  - Load Tools
  - Connectivity Tools
  - Many more
- Pivotal Greenplum Database documentation:
  - http://gpdb.docs.pivotal.io/gpdb-438.html

Pivotal

Thank You

Pivotal

A NEW PLATFORM FOR A NEW ERA