Hannah Roach

4/10/2019

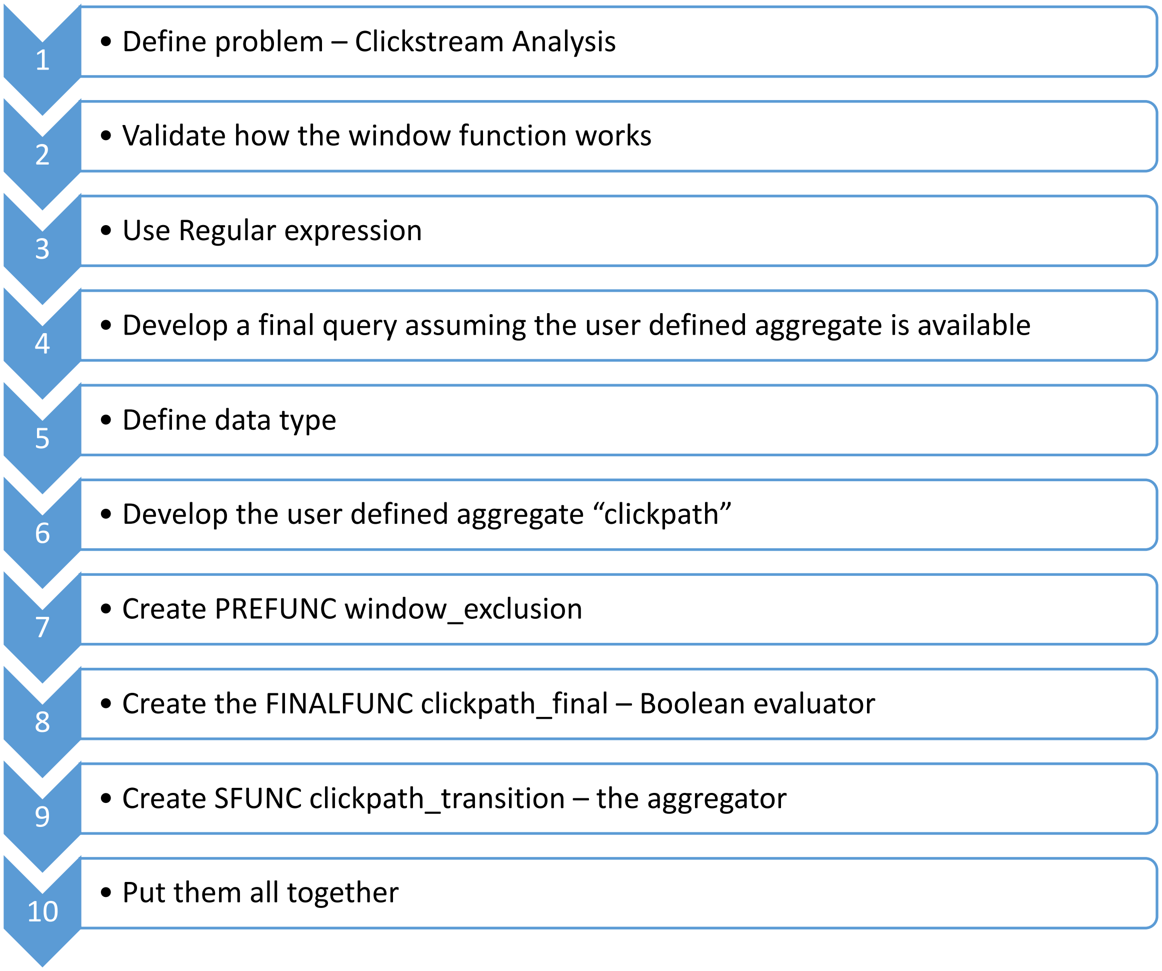
Lab 12: In-database Analytics

Purpose:	This lab is designed to familiarize you and give you practice with the in-database analytics methods. After completing the tasks in this lab you should able to: • Use window functions • Implement user defined aggregates and user defined functions • Use ordered aggregates • Use Regular Expressions (Regex) in SQL for text filtering • Use MADlib functions and plot results from MADlib function outputs

Tasks:	Tasks you'll be completing in this lab include: • Process Clickstream analysis data using window functions, User defined functions, User defined aggregates and regular expressions • Compute median household income using ordered aggregates • Use MADlib functions for logistic regression and direct output to plot the results

Part 1 – In-database analysis of Click-Stream data

Workflow Overview



1
- Define problem – Clickstream Analysis

2
- Validate how the window function works

3
- Use Regular expression

4
- Develop a final query assuming the user defined aggregate is available

5
- Define data type

6
- Develop the user defined aggregate “clickpath”

7
- Create PREFUNC window_exclusion

8
- Create the FINALFUNC clickpath_final – Boolean evaluator

9
- Create SFUNC clickpath_transition – the aggregator

10
- Put them all together

LAB Instructions

Step	Action
1	**Define problem - Clickstream Analysis** Problem Definition: A users' click-stream is defined as the aggregate of all activity a user has through a website via their clicks, derived through the web logs. This has become an important view of the data, it enables insights into typical paths a user takes to navigate a website of interest. Analysis of click-streams can help to improve the usability of the websites, identify hacking attempts on the website, et In this lab, we will be constructing and analyzing click-streams from pre-processed weblog data. You ar provided with data in a database table called "clicks". The table is defined as follows: TABLE clicks(user_id BIGINT, timestamp BIGINT, page_type VARCHAR) Where userid : user session number, page_type : identification of the page visited timestamp : the time of the visit. We want to determine which users: – Start at the home page, – **then** Click on an auction, – **then** View **at least one** help page – **then** Place a bid **In this lab you will connect to "module5indb" database and all the data tables required for this lab ar available in this database. You will be using Putty to complete these steps.**

2

Validate how the window function works

Key in the following code and test how the windows function works:

```
SELECT
  sid
, page_type
, time
, count(*) OVER (prefix) AS seq_length
, count(*) OVER (PARTITION BY sid) AS max_seq_length
FROM
  clicks
WINDOW prefix AS (PARTITION BY sid ORDER BY time ASC)
LIMIT 50
;
```

The SELECT statement selects from table “clicks”, Session_id, Page_type, and the time stamp.

Two standard “count” aggregate functions (which return the count of all records), are also included in the SELECT statement. The first one is defined as “sequence length” and the second one is defined as maximum sequence length.

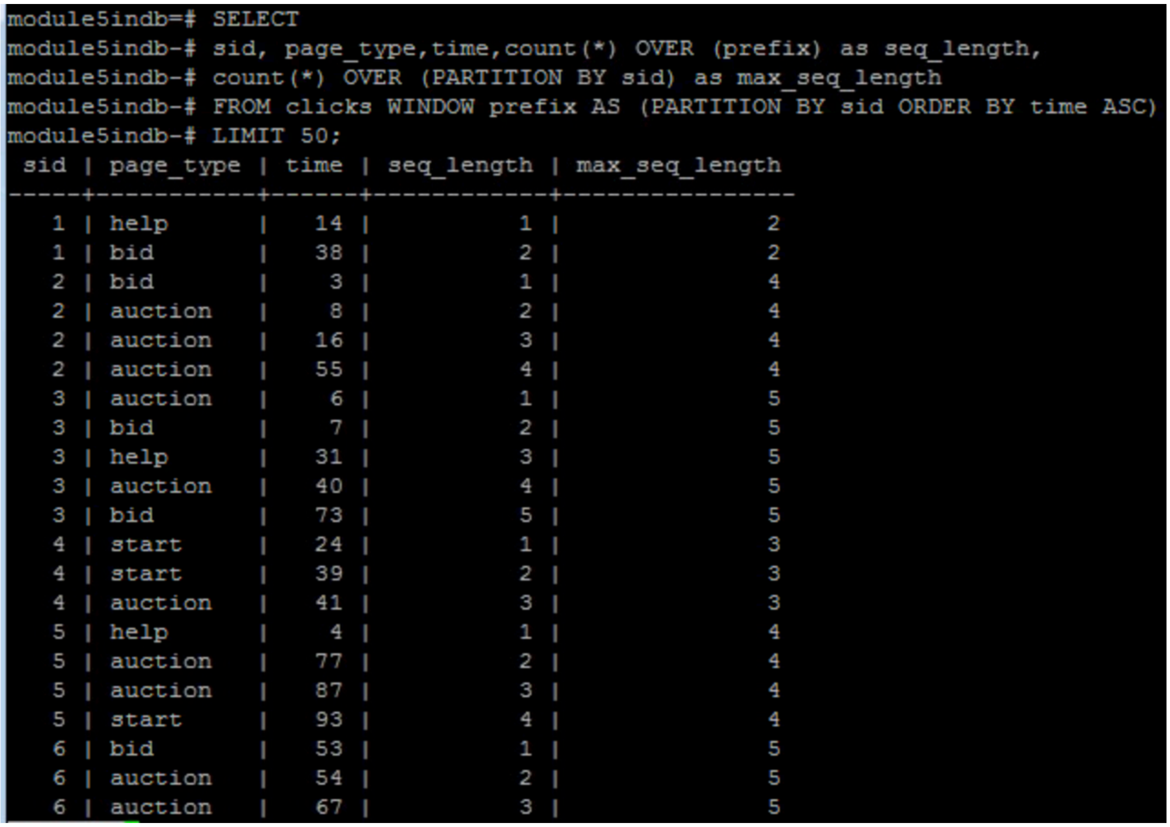
The first “count” aggregate is cumulated over a window defined as “prefix”; “prefix” is partitioned by variable “session id” and ordered by “time” (in a ascending order).

For example if “session_id” = “1” had 10 different clicks at different times, your output for seq_length will be the sequence number of the clicks in the ascending order of time in session_ id = “1”.

If there are 10 clicks that session you will have 10 rows in the output.

The second aggregate is cumulated over the partition defined by session id, the second “count” aggregate in our example will be 10 as there are 10 clicks for “session_id” =1.

Execute the code and observe the results. We have limited the output to 50 rows.
*** SCREENSHOT

Step	Action
	

```
module5indb=# SELECT
module5indb-# sid, page_type,time,count(*) OVER (prefix) as seq_length,
module5indb-# count(*) OVER (PARTITION BY sid) as max_seq_length
module5indb-# FROM clicks WINDOW prefix AS (PARTITION BY sid ORDER BY time ASC)
module5indb-# LIMIT 50;
 sid | page_type | time | seq_length | max_seq_length
-----+-----------+------+------------+----------------
   1 | help      |   14 |          1 |              2
   1 | bid       |   38 |          2 |              2
   2 | bid       |    3 |          1 |              4
   2 | auction   |    8 |          2 |              4
   2 | auction   |   16 |          3 |              4
   2 | auction   |   55 |          4 |              4
   3 | auction   |    6 |          1 |              5
   3 | bid       |    7 |          2 |              5
   3 | help      |   31 |          3 |              5
   3 | auction   |   40 |          4 |              5
   3 | bid       |   73 |          5 |              5
   4 | start     |   24 |          1 |              3
   4 | start     |   39 |          2 |              3
   4 | auction   |   41 |          3 |              3
   5 | help      |    4 |          1 |              4
   5 | auction   |   77 |          2 |              4
   5 | auction   |   87 |          3 |              4
   5 | start     |   93 |          4 |              4
   6 | bid       |   53 |          1 |              5
   6 | auction   |   54 |          2 |              5
   6 | auction   |   67 |          3 |              5
```

3

Use Regular expression

Check through the window defined as “prefix” and determine if the user went through a particular sequence of “page_types”. We want to know if the user (defined by the session_id):

a) Starts at the home page
b) Then clicks on an auction
c) Then views at least one help page
d) Then places a bid

Define an aggregate that will step through the window “prefix” and look at the page types at every record in the window.

If we call our pages with notation S,A,H,B we are looking for a sequence in regular expression terms “^SAH+B”. (defined with a variable “pattern”)

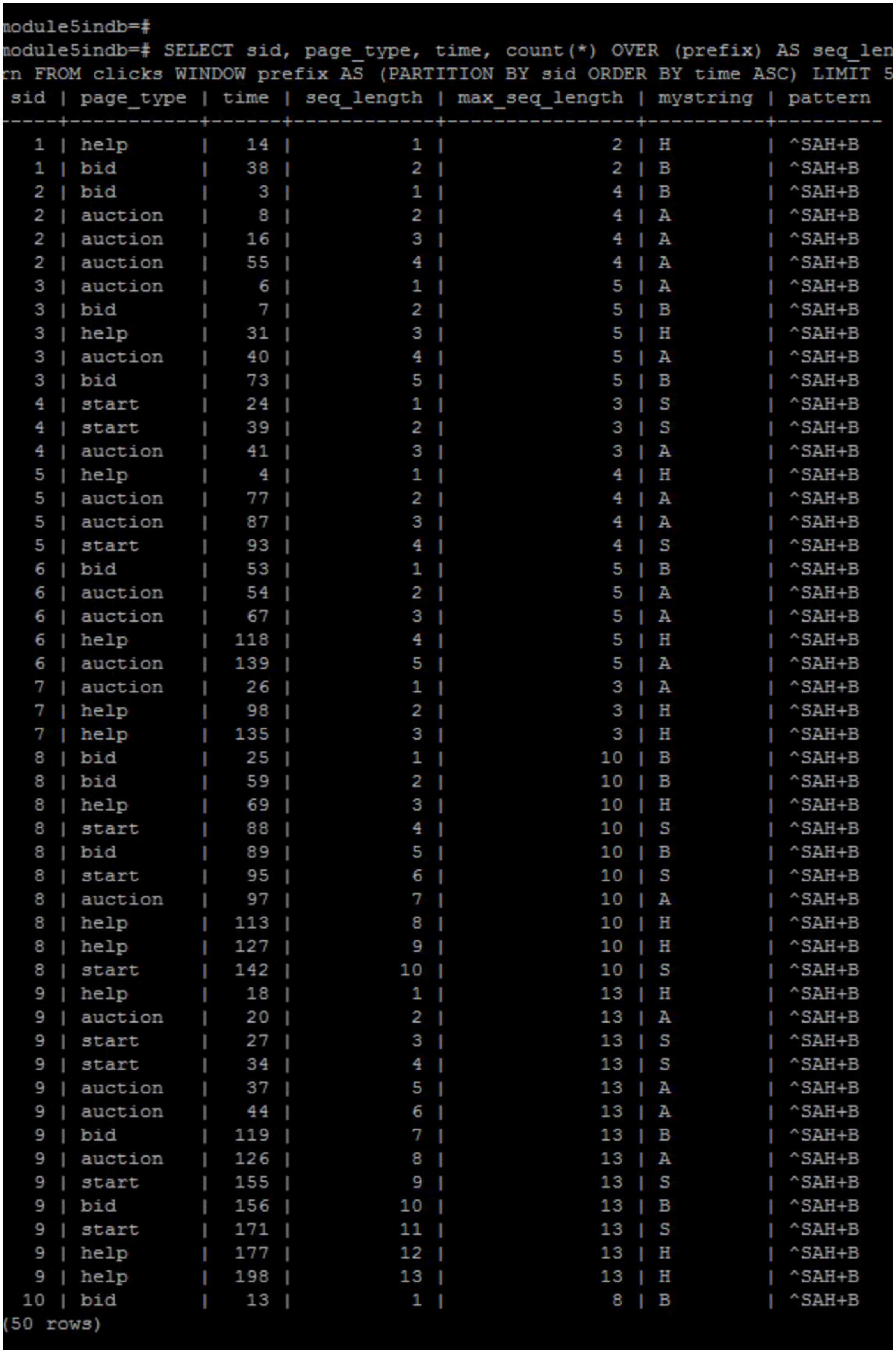
Extract the first character of page_type (use upper case) and build a sequence of the page_type characters and compare this with our regular expression string “^SAB+H”.

The code to perform the above mentioned tasks:

```
SELECT
  sid
, page_type
, time
, count(*) OVER (prefix) AS seq_length
, count(*) OVER (PARTITION BY sid) AS  max_seq_length
, upper(substring(page_type for 1)) AS mystring
, '^SAH+B' AS pattern
FROM
  clicks
WINDOW prefix AS (PARTITION BY sid ORDER BY time ASC)
LIMIT 50
;
```

Review the output.

** SCREENSHOT



```
module5indb=#
module5indb=# SELECT sid, page_type, time, count(*) OVER (prefix) AS seq_len
rn FROM clicks WINDOW prefix AS (PARTITION BY sid ORDER BY time ASC) LIMIT 5
 sid | page_type | time | seq_length | max_seq_length | mystring | pattern
-----+-----------+------+------------+----------------+----------+---------
   1 | help      |   14 |          1 |              2 | H        | ^SAH+B
   1 | bid       |   38 |          2 |              2 | B        | ^SAH+B
   2 | bid       |    3 |          1 |              4 | B        | ^SAH+B
   2 | auction   |    8 |          2 |              4 | A        | ^SAH+B
   2 | auction   |   16 |          3 |              4 | A        | ^SAH+B
   2 | auction   |   55 |          4 |              4 | A        | ^SAH+B
   3 | auction   |    6 |          1 |              5 | A        | ^SAH+B
   3 | bid       |    7 |          2 |              5 | B        | ^SAH+B
   3 | help      |   31 |          3 |              5 | H        | ^SAH+B
   3 | auction   |   40 |          4 |              5 | A        | ^SAH+B
   3 | bid       |   73 |          5 |              5 | B        | ^SAH+B
   4 | start     |   24 |          1 |              3 | S        | ^SAH+B
   4 | start     |   39 |          2 |              3 | S        | ^SAH+B
   4 | auction   |   41 |          3 |              3 | A        | ^SAH+B
   5 | help      |    4 |          1 |              4 | H        | ^SAH+B
   5 | auction   |   77 |          2 |              4 | A        | ^SAH+B
   5 | auction   |   87 |          3 |              4 | A        | ^SAH+B
   5 | start     |   93 |          4 |              4 | S        | ^SAH+B
   6 | bid       |   53 |          1 |              5 | B        | ^SAH+B
   6 | auction   |   54 |          2 |              5 | A        | ^SAH+B
   6 | auction   |   67 |          3 |              5 | A        | ^SAH+B
   6 | help      |  118 |          4 |              5 | H        | ^SAH+B
   6 | auction   |  139 |          5 |              5 | A        | ^SAH+B
   7 | auction   |   26 |          1 |              3 | A        | ^SAH+B
   7 | help      |   98 |          2 |              3 | H        | ^SAH+B
   7 | help      |  135 |          3 |              3 | H        | ^SAH+B
   8 | bid       |   25 |          1 |             10 | B        | ^SAH+B
   8 | bid       |   59 |          2 |             10 | B        | ^SAH+B
   8 | help      |   69 |          3 |             10 | H        | ^SAH+B
   8 | start     |   88 |          4 |             10 | S        | ^SAH+B
   8 | bid       |   89 |          5 |             10 | B        | ^SAH+B
   8 | start     |   95 |          6 |             10 | S        | ^SAH+B
   8 | auction   |   97 |          7 |             10 | A        | ^SAH+B
   8 | help      |  113 |          8 |             10 | H        | ^SAH+B
   8 | help      |  127 |          9 |             10 | H        | ^SAH+B
   8 | start     |  142 |         10 |             10 | S        | ^SAH+B
   9 | help      |   18 |          1 |             13 | H        | ^SAH+B
   9 | auction   |   20 |          2 |             13 | A        | ^SAH+B
   9 | start     |   27 |          3 |             13 | S        | ^SAH+B
   9 | start     |   34 |          4 |             13 | S        | ^SAH+B
   9 | auction   |   37 |          5 |             13 | A        | ^SAH+B
   9 | auction   |   44 |          6 |             13 | A        | ^SAH+B
   9 | bid       |  119 |          7 |             13 | B        | ^SAH+B
   9 | auction   |  126 |          8 |             13 | A        | ^SAH+B
   9 | start     |  155 |          9 |             13 | S        | ^SAH+B
   9 | bid       |  156 |         10 |             13 | B        | ^SAH+B
   9 | start     |  171 |         11 |             13 | S        | ^SAH+B
   9 | help      |  177 |         12 |             13 | H        | ^SAH+B
   9 | help      |  198 |         13 |             13 | H        | ^SAH+B
  10 | bid       |   13 |          1 |              8 | B        | ^SAH+B
(50 rows)
```

Step	Action

4

Develop a final query assuming the user defined aggregate is available

The output of the column is the first character of “page id”. As you step through each time stamp of th “preview” window, aggregate the first characters at each pass.
This aggregated character set is compared with the “pattern” “^SAH+B”.

Write a user defined aggregate that will accumulate the text string on each step it traverses in the window and return a Boolean value “true” or “false” based on the match with the pattern.

Call this function “clickpath” and the arguments for this function are

- the upper cased first character of the page_type and
- the regular expression “pattern”

```
clickpath(upper(substring(page_type for 1)), '^SAH+B' )
```

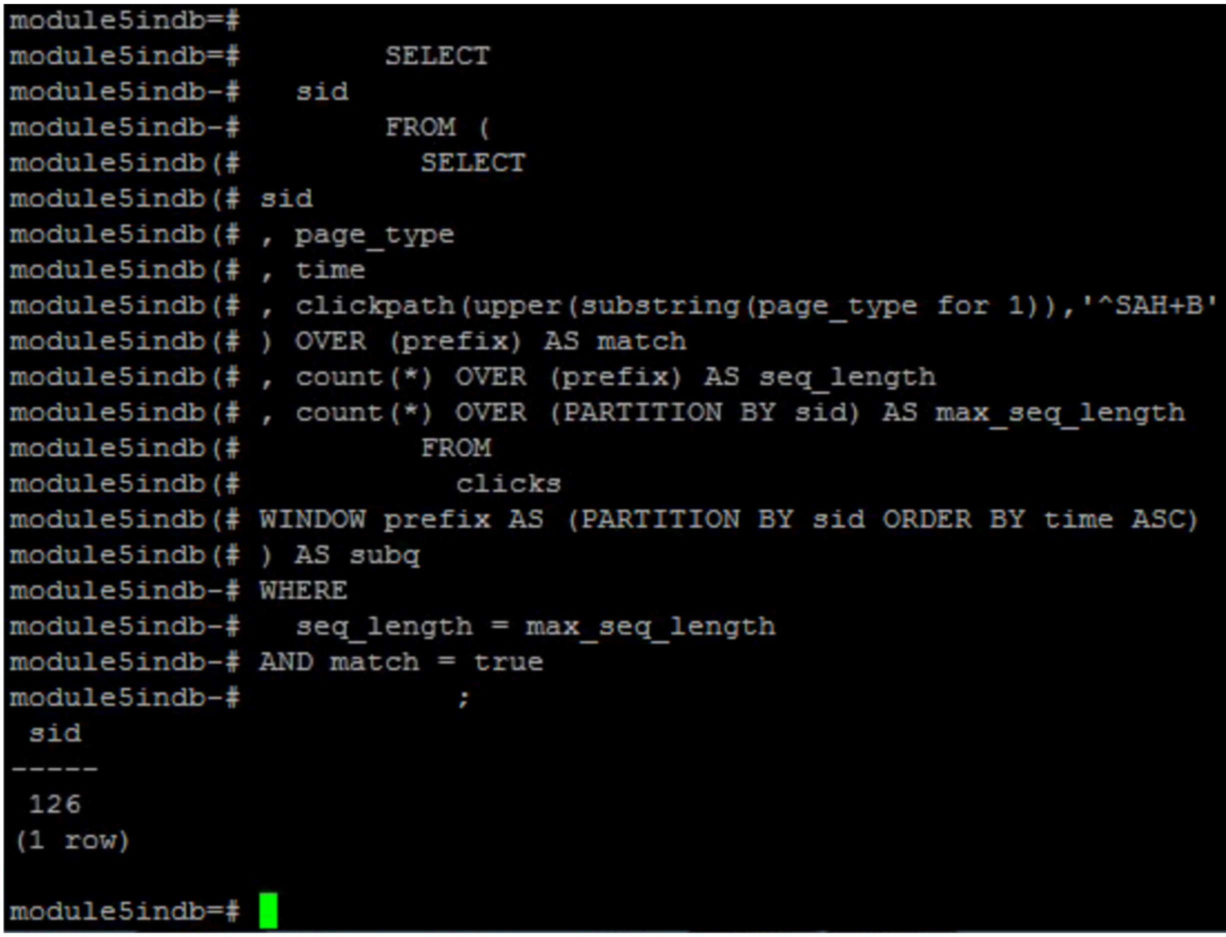
This function should work as an aggregate over the window “prefix” accumulating the first character an determining the Boolean value of match.

Our final query code (assuming clickpath works the way it is intended) will be:

```
SELECT
  sid
FROM (
  SELECT
    sid
  , page_type
  , time
  , clickpath(upper(substring(page_type for 1)),'^SAH+B'
  ) OVER (prefix) AS match
  , count(*) OVER (prefix) AS seq_length
  , count(*) OVER (PARTITION BY sid) AS max_seq_length
  FROM
    clicks
  WINDOW prefix AS (PARTITION BY sid ORDER BY time ASC)
  ) AS subq
  WHERE
    seq_length = max_seq_length
    AND match = true
  ;
```

Which session ID (sid) matches the desired patten (start at the Home Page, click on the Auction, view at least one Help page, and places Bid)

Step	Action



```
module5indb=#
module5indb=#         SELECT
module5indb-#    sid
module5indb-#         FROM (
module5indb(#           SELECT
module5indb(# sid
module5indb(# , page_type
module5indb(# , time
module5indb(# , clickpath(upper(substring(page_type for 1)),'^SAH+B'
module5indb(# ) OVER (prefix) AS match
module5indb(# , count(*) OVER (prefix) AS seq_length
module5indb(# , count(*) OVER (PARTITION BY sid) AS max_seq_length
module5indb(#           FROM
module5indb(#             clicks
module5indb(# WINDOW prefix AS (PARTITION BY sid ORDER BY time ASC)
module5indb(# ) AS subq
module5indb-# WHERE
module5indb-#    seq_length = max_seq_length
module5indb-# AND match = true
module5indb-#             ;
 sid
-----
 126
(1 row)

module5indb=#
```

5

<u>Define data type</u>

Define a composite data type that you will use with the aggregation function.
Our composite data type will consists of

- the sequence we are aggregating and
- a regular expression “pattern” (which does not change) that we will use for comparison.

Create data type with the following code:

```
DROP TYPE IF EXISTS clickstream_state CASCADE;
CREATE TYPE clickstream_state AS (
  sequence VARCHAR
, pattern VARCHAR
);
```

Step | **Action**

6

Develop the user defined aggregate “clickpath”

There are two major functions of “clickpath”

- It should aggregate the characters (transition function that aggregates)
- It should compare and return a Boolean function (the final function that returns the Boolean value)

Key in the following code:

```
DROP AGGREGATE IF EXISTS clickpath(
  /* Symbol */ CHAR
, /* regex */ TEXT
  );
CREATE AGGREGATE clickpath(
  /* Symbol */ CHAR
, /* regex */ TEXT)
(
    STYPE = clickstream_state,
    SFUNC = clickpath_transition,
    FINALFUNC = clickpath_final,
    PREFUNC = window_exclusion
);
```

Note:

The STYPE is the data type we defined in step 5.

We need to create two functions (detailed in steps 8 and 9 later):

- clickpath_transition (the aggregator) and
- clickpath_final (the Boolean evaluator)

Notice that we also defined a PREFUNC, a function required to enable the function clickpath to be called as a window function.

7

Create PREFUNC window_exclusion:

```
CREATE OR REPLACE FUNCTION window_exclusion(clickstream_state,
clickstream_state)
RETURNS clickstream_state AS $$
BEGIN
    RAISE EXCEPTION 'aggregate may only be called from a window
function';
END;
$$ LANGUAGE PLPGSQL STRICT;
```

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>8</td><td>

Create the FINALFUNC clickpath_final - Boolean evaluator:

The Boolean evaluator is the simpler of the two remaining functions.

```
CREATE OR REPLACE FUNCTION clickpath_final(state
clickstream_state)
RETURNS BOOLEAN AS $$
    SELECT $1.sequence ~ $1.pattern;
$$ LANGUAGE SQL STRICT;
```

The sequence and the pattern are matched and the Boolean value is returned. $1 refers to the first and the only argument in the function call. Recall the composite data type we created has both sequence and pattern.

</td></tr>
<tr><td>9</td><td>

Create SFUNC clickpath_transition – the aggregator
The next and the last function to define is the aggregator. This function has three arguments.

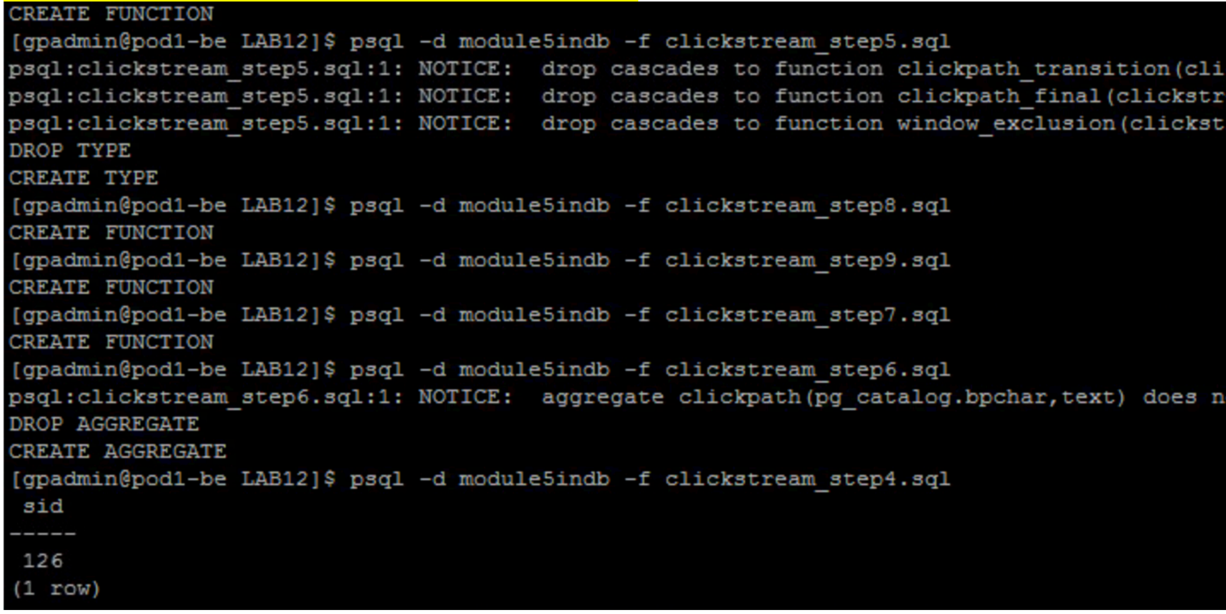
- The “state” which aggregates with every step,
- The “symbol”, the character we read in from the current row
- The pattern to match

When you step into a new window, the “state” will be NULL and it will take in the first character. As we step through each row within the window the aggregation will be carried out.

Code the function as follows:

```
CREATE OR REPLACE FUNCTION clickpath_transition(
  state clickstream_state, symbol CHAR(1), pattern VARCHAR)
RETURNS clickstream_state AS $$
    SELECT CASE
        WHEN $1 IS NULL THEN ($2, $3)::clickstream_state
        ELSE ($1.sequence || $2, $3)::clickstream_state
    END;
$$ LANGUAGE SQL CALLED ON NULL INPUT;
```

</td></tr>
</table>

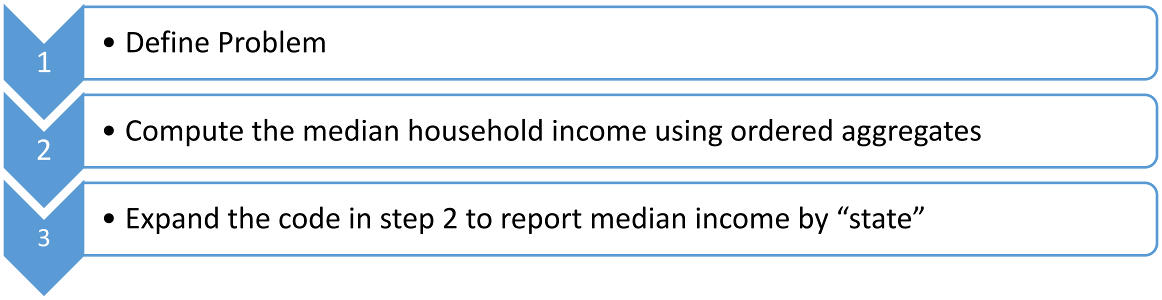
<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>10</td><td>
<b><u>Put them all together</u>:</b>
<u>Check your results</u>
<ol>
<li>Start with the definition of data type (step 5)</li>
<li>Code the three functions SFUNC, FINALFUNC and PREFUNC (Steps 8,9,7)</li>
<li>Complete the user defined aggregate (step6)</li>
<li>Run the query (step4)3</li>
</ol>
<b>Do you get the same SID ID as you did in step 4? YES</b>

<pre>
CREATE FUNCTION
[gpadmin@pod1-be LAB12]$ psql -d module5indb -f clickstream_step5.sql
psql:clickstream_step5.sql:1: NOTICE: drop cascades to function clickpath_transition(cli
psql:clickstream_step5.sql:1: NOTICE: drop cascades to function clickpath_final(clickstr
psql:clickstream_step5.sql:1: NOTICE: drop cascades to function window_exclusion(clickst
DROP TYPE
CREATE TYPE
[gpadmin@pod1-be LAB12]$ psql -d module5indb -f clickstream_step8.sql
CREATE FUNCTION
[gpadmin@pod1-be LAB12]$ psql -d module5indb -f clickstream_step9.sql
CREATE FUNCTION
[gpadmin@pod1-be LAB12]$ psql -d module5indb -f clickstream_step7.sql
CREATE FUNCTION
[gpadmin@pod1-be LAB12]$ psql -d module5indb -f clickstream_step6.sql
psql:clickstream_step6.sql:1: NOTICE: aggregate clickpath(pg_catalog.bpchar,text) does n
DROP AGGREGATE
CREATE AGGREGATE
[gpadmin@pod1-be LAB12]$ psql -d module5indb -f clickstream_step4.sql
 sid

 126
(1 row)
</pre>
The segments of this code are available in <b>/home/gpadmin/LAB12/clickstream_step*.sql</b>
(* represents the steps in the document).
</td></tr>
</table>

End of Lab Exercise

Part 2 – In-database computation of Median with Ordered Aggregates

Workflow Overview



1

- Define Problem

2

- Compute the median household income using ordered aggregates

3

- Expand the code in step 2 to report median income by “state”

LAB Instructions

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>

Define Problem:

Use the housing table in training2 database (census) to compute the median household income for each state.

</td></tr>
<tr><td>2</td><td>

Compute the median household income using ordered aggregates

Use ordered aggregates for the computation of median household income. Code suggestion:

```
SELECT
  ( arr[ length/2 + 1 ] + arr[ (length + 1)/2 ] ) / 2.0 AS
median_income
FROM(
  SELECT
    array_agg(hinc ORDER BY hinc) AS arr
  , count(*) AS length
  FROM
    housing
 ) AS q
;
```

What is the overall median household income in the US?

</td></tr>
</table>

Step	Action
3	**Expand the code in step 2 to report median income by “state”:** Execute the following code:

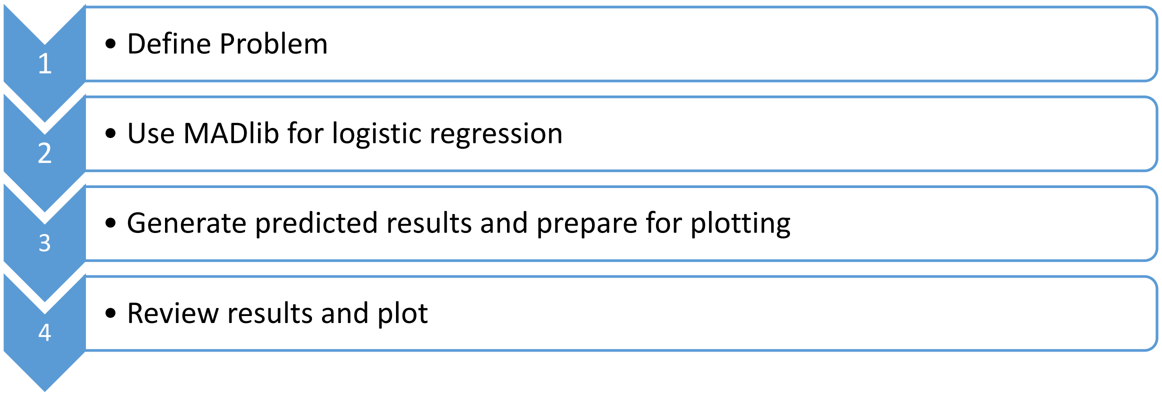
```
SELECT
  f.name
, ( arr[ length/2 + 1 ] + arr[ (length + 1)/2 ] ) / 2.0 AS
median_income
FROM(
  SELECT
    state AS s
  , array_agg(hinc ORDER BY hinc) AS  arr
  , count(*) AS length
  FROM
    housing
  GROUP BY
    state
 ) AS q
JOIN
  fips f
ON
  s = f.code
ORDER BY
  f.name
;
```

What is the median income of Massachusetts and Alaska?

End of Lab Exercise

Part 3: Logistic Regression with MADlib

Workflow Overview



1. Define Problem
2. Use MADlib for logistic regression
3. Generate predicted results and prepare for plotting
4. Review results and plot

LAB Instructions

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>

Define Problem:

In this exercise you will use the MADlib function for logistic regression and generate the model and plot the predicted results. Synthetic data is available in the table “artificiallogreg” in database “module5indb”

</td></tr>
<tr><td>2</td><td>

Use MADlib for logistic regression

Execute the following code to generate the model and store the results in a table “logr_coef”

```
DROP TABLE IF EXISTS logr_coef;
CREATE TABLE logr_coef AS
  SELECT 0::INT AS bla
, NULL::FLOAT8[] AS coef
DISTRIBUTED BY (bla)
;

UPDATE logr_coef SET coef = (SELECT coef FROM
madlib.logregr('artificiallogreg', 'y', 'x', 20, 'irls',
0.001) AS coef)
;
```

</td></tr>
<tr><td>3</td><td>

Generate predicted results and prepare for plotting

Generate the predicted results; organize them in ascending order of value of x. Pipe the results using meta commands “\o” to a file called “graphics.txt” that we can use to plot in the next step:

```
\a
\o graphics.txt
SELECT
  DISTINCT rank::FLOAT8/total_count AS x
, count::FLOAT8/total_true AS y
FROM (
  SELECT
    y
  , rank() OVER (ORDER BY prediction DESC)
  , count(*) OVER () total_count
  , count(*) FILTER (WHERE y = TRUE) OVER (ORDER BY
prediction DESC)
  , count(*) FILTER (WHERE y = TRUE) OVER () AS
total_true
```

</td></tr>
</table>

Step	Action
3 cont'd	FROM (SELECT r.* , 1. / (1. + exp(-dotProduct(r.x, c.coef))) AS prediction FROM artificiallogreg AS r CROSS JOIN logr_coef as c) q) p ; \o
4	**View the graphic.txt file and share a screenshot.**

End of Lab Exercise