# 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

### 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, as 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, etc. In this lab, we will be constructing and analyzing click-streams from pre-processed weblog data. You are 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 are 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 |
| 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):   * 1. Starts at the home page   2. Then clicks on an auction   3. Then views at least one help page   4. 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 |
| 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 the “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 and 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)** |
| 5 | **Define data type**  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**  **);** |
| 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;** |
| 8 | **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. |
| 9 | **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;** |
| 10 | **Put them all together**:  Check your results   1. Start with the definition of data type (step 5) 2. Code the three functions SFUNC, FINALFUNC and PREFUNC (Steps 8,9,7) 3. Complete the user defined aggregate (step6) 4. Run the query (step4)3   **Do you get the same SID ID as you did in step 4? YES**    The segments of this code are available in **/home/gpadmin/LAB12/clickstream\_step\*.sql**  (\* represents the steps in the document). |

*End of Lab Exercise*

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

### Workflow Overview

### LAB Instructions

| **Step** | **Action** |
| --- | --- |
| 1 | **Define Problem:**  Use the housing table in training2 database (census) to compute the median household income for each state. |
| 2 | **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? |
| 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

### LAB Instructions

| **Step** | **Action** |
| --- | --- |
| 1 | **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” |
| 2 | **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)**  **;** |
| 3 | **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** |
| 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*