Module 2: Path & Pattern Detection with nPath

Teradata Vantage Analytics Workshop

BASIC

Objectives

After completing this module, you will be able to:

- Describe what the **nPath** function does
- Describe typical use cases for **nPath**
- Write **nPath** queries
- Interpret the output of **nPath** queries

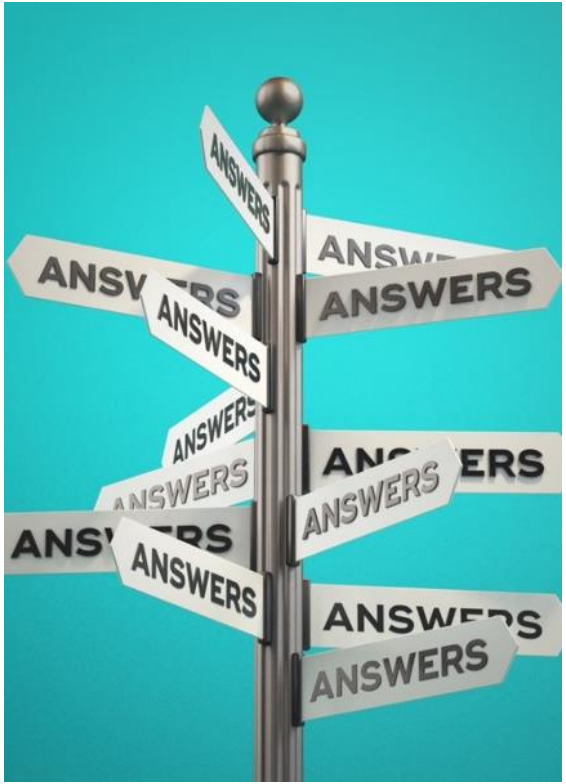
For more info go to docs.teradata.com click Teradata Vantage, download: Teradata Vantage Analytic Function Reference Guide

Topics

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
- Symbols
- Mode
- Pattern with Symbols
- Result
- Daisy Chaining, Multiple Input, & Filter Argument
- Vantage Analyst
- Hackathon & Review

Current Topic – Background Information

- **Background Information**
  - **Description**
  - **Use Cases**
  - **Syntax**
  - **Input Data**
  - **Required Arguments**
- Symbols
- Mode
- Pattern with Symbols
- Result
- Daisy Chaining, Multiple Input, & Filter Argument
- Vantage Analyst
- Hackathon & Review



Description

What is nPath?

- Function designed for time-series sequence analysis of data
- Links an outcome with a preceding path

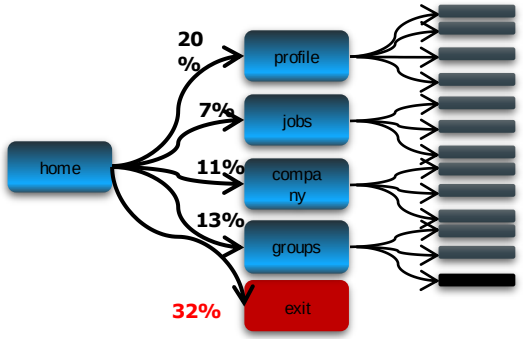
Benefits

- *Pattern detection can be completed in a single pass* over the data
- Allows you to understand relationships across rows of data
- SQL's ordered-data limitations that require either complex, multi-pass SQL or custom UDFs for each analysis

Example use cases:

- Web analytics (clickstream, Golden Path)
- Complex Marketing revenue paths
- Granular product & process analysis (A/B)
- Granular pattern detection (fraud, QA,..)

nPath Website Path Analysis



Complete Application:

- ***n*Path** identifies path patterns and exit points
- The **Sessionize** function is used prepare the input data for **nPath** analysis.
- In the example above, we are viewing which paths users take once they've landed on our Home page. Note that 32% of Home page visits end up in the users' exiting the website altogether.

nPath Use Cases

Some examples of how **nPath** can be used follow:

- A Retailer wishes to analyze Web site click data, to identify paths that lead to sales over a specified amount
- A Manufacturer analyzes sensor data FROM industrial processes, to identify paths to poor product quality
- A Healthcare provider analyzes healthcare records of individual patients, to identify paths that indicate that patients are at risk of developing conditions such as heart disease or diabetes
- A Financial institution reviews financial data for individuals, to identify paths that provide information about credit or fraud risks

nPath Workflow

- **Input Tables(s):** Data is read FROM specified input tables, views, or queries
- **nPath:** The following arguments are specified when the function is invoked
  - **Mode (overlapping or nonoverlapping)**
  - **Pattern to match**
  - **Symbols to use**
  - **[Optional] Filters to apply**
  - **Results to output**
- **Output table:** Data is written to an output table (or displayed to Console)

nPath Syntax


```
SELECT * FROM nPath
(ON { table | view | (query) }
PARTITION BY partition_column
ORDER BY order_column[ ASC | DESC ]
[ ON { table | view | (query) }
[ PARTITION BY partition_column | DIMENSION ] ORDER BY column [ ASC | DESC ]]
USING
Mode ({ OVERLAPPING | NONOVERLAPPING })
Symbols ({ col_expr = symbol_predicate AS symbol } [,...])
Pattern ('pattern')
[ Filter (filter_expression [,...]) ]
Result ({ aggregate_function (col_expr OF symbol) AS alias_1 }[,...])
) AS dt;
```

Lab 1: nPath Simple Query

Input

```
SELECT * FROM borre_z
ORDER BY ts;
```

	user_id	event	ts
1	1	a	2017-01-01 13:21:01.000000
2	1	a	2017-01-01 13:21:02.000000
3	1	a	2017-01-01 13:21:03.000000
4	1	a	2017-01-01 13:21:04.000000

nPath Query

```
SELECT * FROM nPath
(ON borre_z
 PARTITION BY user_id
 ORDER BY ts
 USING
 Mode (NONOVERLAPPING)
 Symbols (event = 'a' as X)
 Pattern ('X.X')
 Result (Accumulate (event OF X) AS x_pattern)
) AS dt;
```

- On the following pages, we will discuss the required arguments for **nPath**
- For each required argument, we will discuss the implications of our specifications using the simple **nPath** query to the right as the foundation

nPath Results

	x_pattern
1	[a, a]
2	[a, a]

nPath Input Data


```
SELECT * FROM nPath@coprocessor (
ON { table | view | (query) } PARTITION BY partition_column ORDER BY order_column
[ ASC | DESC ]
[ ON { table | view | (query) }
[ PARTITION BY partition_column | DIMENSION ] ORDER BY order_column [ ASC | DESC ]
][...]
```

Here, we specify our input data:

- The **FROM** keyword is followed by **nPath.** This invokes the **nPath** function
- The **ON** keyword is followed by our input data (**borre_z**)
- We **PARTITION BY** user_id in this example and **ORDER BY** ts

nPath Query

```
SELECT * FROM nPath
(ON borre_z
 PARTITION BY user_id
 ORDER BY ts
 USING
 Mode (NONOVERLAPPING)
 Pattern ('X.X')
 Symbols (event = 'a' as X)
 Result (Accumulate (event OF X) AS x_pattern)
) AS dt;
```

nPath Input Data (cont.)


```
… ON (SELECT * FROM WEBCLICKS)
  PARTITION BY user_id, sessionid
  ORDER BY datestamp …
```

AMP 0

Partition = User_Id 1 Session_Id 0

User_Id	Session_Id	Date stamp	Other Columns
1	0	2011-01-02 02:15:00	Home page, etc,
1	0	2011-01-02 02:16:00	Product page, etc
1	0	2011-01-02 02:17:00	Profile page, etc
1	0	2011-01-02 02:18:00	Watch page, etc
1	0	2011-01-02 02:19:00	Logout page, etc

AMP 1

Partition = User_Id 1 Session_Id 1

User_Id	Session_Id	Date stamp	Other Columns
1	1	2011-01-11 09:27:00	Home page, etc,
1	1	2011-01-11 09:28:00	Product page, etc
1	1	2011-01-11 009:30:00	Product page, etc
1	1	2011-01-11 09:32:00	Bid, etc

AMP 2

Partition = User_Id 97 Session_Id 0

User_Id	Session_Id	Date stamp	Other Columns
97	0	2011-03-13 07:17:00	Home page, etc,
97	0	2011-03:13 07:18:00	Coupon page, etc
97	0	2011-03:13 07:19:00	Customer support page, etc
97	0	2011-03-13 07:21:00	Sell page, etc
97	0	2011-03-13 07:23:00	History page, etc

PARTITION BY groups rows with like values together.
ORDER BY then sorts each partition according to our specifications.
Clicks of each User_Id/Session_id are now sequenced on AMP

nPath Input Data (cont.)

- **ON expression**
  - The input Table, View, or Query
- **PARTITION BY expression [,…]**
  - The attribute(s) by which the rows are grouped
  - Identifies entity of interest – such as user_id, product_id, etc.
- **ORDER BY expression [ASC|DESC] [,…]**
  - The expression by which the rows within each partition are ordered
  - Typically, a date/time field, but can be any sequence attribute

Selecting all rows from savings

```
SELECT * FROM nPath
(ON savings
 PARTITION BY cust
 ORDER BY ts
……….
```

Selecting a subset of rows from savings

```
SELECT * FROM nPath
(ON (SELECT * FROM savings
     WHERE amt > 0) as t1
 PARTITION BY cust
 ORDER BY ts
……….
```

nPath Required Arguments: Mode


```
USING
Mode ({ OVERLAPPING | NONOVERLAPPING })
Pattern ('pattern')
Symbols ({ col_expr = symbol_predicate AS symbol } [,...])
[ Filter (filter_expression [,...]) ]
Result ({ aggregate_function (col_expr OF symbol) AS alias_1 }[,...])
) AS alias_2;
```

Here, we specify the pattern-matching mode. There are two flavors of this:

- **OVERLAPPING:** Find every occurrence of pattern in partition, regardless of whether it is part of a previously found match. One row can match multiple symbols in a given matched pattern
- **NONOVERLAPPING:** Start next pattern search at row that follows last pattern match

```
SELECT * FROM nPath
(ON borre_z
 PARTITION BY user_id ORDER BY ts
 USING
 Mode (NONOVERLAPPING)
 Symbols (event = 'a' as X)
 Pattern ('X.X')
 Result (Accumulate (event OF X) AS event1)
) AS dt;
```

nPath Required Arguments: Symbols


```
USING
Mode ({ OVERLAPPING | NONOVERLAPPING })
Pattern ('pattern')
Symbols ({ col_expr = symbol_predicate AS symbol } [,...])
[ Filter (filter_expression [,...]) ]
Result ({ aggregate_function (col_expr OF symbol) AS alias_1 }[,...])
) AS alias_2;
```

Here, we specify the **Symbols** that appear in the values of the **Pattern** and **Result** arguments

- The *col_expr* is an expression whose value is a column name, symbol is any valid identifier, and *symbol_predicate* is a SQL predicate (often a column name)
- You can think of **Symbols** as the 'aliases' that you will be defining for use in the **Pattern** and **Result** portions of the query

```
SELECT * FROM nPath
(ON borre_z
 PARTITION BY user_id
 ORDER BY ts
 USING
 Mode (NONOVERLAPPING)
 Symbols (event = 'a' AS X)
 Pattern ('X.X')
 Result (Accumulate (event OF X) AS event1)
) AS dt;
```

nPath Required Arguments: Pattern


```
USING
Mode ({ OVERLAPPING | NONOVERLAPPING })
Pattern ('pattern')
Symbols ({ col_expr = symbol_predicate AS symbol } [,...])
[ Filter (filter_expression [,...]) ]
Result ({ aggregate_function (col_expr OF symbol) AS alias_1 }[,...])
) AS alias_2;
```

Here, we specify the **Pattern** the function searches for

- We compose **Pattern** with symbols (which we define in **Symbols** argument), operators, and parentheses. Here, we are searching for Symbol X *followed by* X (which means event 'a' *followed by* event 'a')
- When patterns have multiple operators, the function applies them in order of precedence, and applies operators of equal precedence FROM left to right. To force the function to evaluate a subpattern first, enclose it in parentheses

```
SELECT * FROM nPath
(ON borre_z
 PARTITION BY user_id
 ORDER BY ts
 USING
 Mode (NONOVERLAPPING)
 Symbols (event = 'a' as X)
 Pattern ('X.X')
 Result (Accumulate(event OF X) AS event1)
) AS dt;
```

nPath Required Arguments: Result


```
USING
Mode ({ OVERLAPPING | NONOVERLAPPING })
Pattern ('pattern')
Symbols ({ col_expr = symbol_predicate AS symbol } [,...])
[ Filter (filter_expression [,...]) ]
Result ({ aggregate_function (col_expr OF symbol) AS alias_1 }[,...])
) AS alias_2;
```

Here, we specify the Output columns

- The *col_expr* is an expression whose value is a column name; it specifies the values to retrieve FROM the matched rows. The function applies aggregate_function to these values
- The function evaluates this argument once for every matched pattern in the partition (that is, it outputs one row for each pattern match)

```
SELECT * FROM nPath
(ON borre_z
 PARTITION BY user_id
 ORDER BY ts
 USING
 Mode (NONOVERLAPPING)
 Symbols (event = 'a' as X)
 Pattern ('X.X')
 Result (Accumulate (event OF X) AS event1)
) AS dt;
```

Lab 2: Changing the Order of Required Function Arguments Doesn't Make a Difference

- You can change the order of arguments that appear after **USING** as desired
- For example, you may find it more logical to define **Symbols** right away
- All of the following queries will return the same answer-set

```
SELECT * FROM nPath
(ON borre_z
 PARTITION BY user_id
 ORDER BY ts
 USING
 Mode (NONOVERLAPPING)
 Symbols (event = 'a' as X)
 Pattern ('X.X')
 Result (Accumulate (event OF X)
AS x_pattern)) AS dt;
```

	x_pattern
1	[a, a]
2	[a, a]

```
SELECT * FROM nPath
(ON borre_z
 PARTITION BY user_id
 ORDER BY ts
 USING
 Pattern ('X.X')
 Mode (NONOVERLAPPING)
 Symbols (event = 'a' as X)
 Result (Accumulate (event OF X)
AS x_pattern)) AS dt;
```

	x_pattern
1	[a, a]
2	[a, a]

```
SELECT * FROM nPath
(ON borre_z
 PARTITION BY user_id
 ORDER BY ts
 USING
 Symbols (event = 'a' as X)
 Mode (NONOVERLAPPING)
 Pattern ('X.X')
 Result (Accumulate (event OF X)
AS x_pattern)) AS dt;
```

	x_pattern
1	[a, a]
2	[a, a]

Current Topic – Symbols

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
- **Symbols**
- Mode
- Pattern with Symbols
- Result
- Daisy Chaining, Multiple Input, & Filter Argument
- Vantage Analyst
- Hackathon & Review

Symbols Predicates

Following labs will be using following Predicates in the SYMBOL arguments

Syntax

```
SYMBOLS ([column] operator [value] as Alias, … ) and/or
          TRUE as Alias (ie: any row)
```

Predicates	Description
TRUE AS <alias>	Symbol can match any row (often used for exploratory queries)
% _	**%** (any # of char) **_** (positional) : Used for character comparisons
LIKE NOT LIKE	Case sensitive text comparison
ILIKE NOT ILIKE	Case insensitive text comparison
LAG LEAD	Compare current row to prior/next row(s)
=, <, >, <=, >= <>	Numeric comparisons

Will show more examples of **SYMBOLS** Predicate later when discussing **PATTERN** argument

Lab 3a: Input Table for Symbols Labs

- Recall that the **Symbols** argument allows us to define the aliases that we wish to use in the **Pattern** and **Results** arguments
- The next few pages will walk through some very straightforward examples that should illustrate how **Symbols** work
- For all examples, assume four-row table appearing below
- For all examples, we will be searching for a product of ***apple*** *followed by* ***banana***, using whatever **Symbols** we may have decided to define
- In our dataset, the only pattern match will be rows 1 (apple) and 2 (banana)

```
SELECT *
FROM borre_food
ORDER BY event_id;
```

	user_name	event_id	product
1	tom	0	apple
2	tom	1	banana
3	tom	2	cherry
4	tom	3	date

Lab 3b: Symbols Example

Query: Accumulate path of 'apple' followed by 'banana' and count frequency

```
SELECT products_accumulate, count(*)
FROM nPath
(ON borre_food
 PARTITION BY user_name
 ORDER BY event_id
 USING
 Mode (NONOVERLAPPING)
 Symbols(product = 'apple' as a,
         product = 'banana' as b)
 Pattern ('a.b')
 Result (ACCUMULATE (product OF ANY (a,b)
         DELIMITER '*')
 AS products_accumulate)
) AS dt
GROUP BY products_accumulate;
```

- Here, we are defining **Symbols** of ***a*** *for apple* and ***b*** *for banana*
- Our defined **Symbols** are used in the **Pattern** and **Result** arguments

Input

	user_name	event_id	product
1	tom	0	apple
2	tom	1	banana
3	tom	2	cherry
4	tom	3	date

Output

	products_accumulate	Count(*)
1	[apple*banana]	1

Note Aggregation after function executes using GROUP BY along with the aggregation function in the SELECT clause

Current Topic – Mode

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
- Symbols
- **Mode**
- Pattern with Symbols
- Result
- Daisy Chaining, Multiple Input, & Filter Argument
- Vantage Analyst
- Hackathon and Review

Lab 4a: View the Data

- **Mode** argument can have a value of **NONOVERLAPPING** or **OVERLAPPING**
- The next few pages will walk through how the **Mode** value that you specify will impact the answer-set
- For the example, assume a simple input table with two columns and four rows

```
SHOW TABLE matchup;

SELECT * FROM matchup
ORDER BY c2, c1;
```

```
CREATE MULTISET TABLE matchup ,FALLBACK ,
      NO BEFORE JOURNAL,
      NO AFTER JOURNAL,
      CHECKSUM = DEFAULT,
      DEFAULT MERGEBLOCKRATIO,
      MAP = TD_MAP1
      (
       c1 INTEGER,
       c2 VARCHAR(30) CHARACTER SET LATIN NOT CASESPECIFIC)
PRIMARY INDEX ( c1 );
```

	c1	c2
1	1	A
2	2	A
3	3	A
4	4	A

Lab 4b: Mode(Nonoverlapping)

PATTERN(A.A) = Search for A followed by A

In **NONOVERLAPPING** match mode, **nPath** begins the next pattern search at the row that follows the last row that was part of the previous **PATTERN** match. In this example, the next pattern match starts at row 3

```
SELECT * FROM nPath
(ON matchup
 PARTITION BY c2
 ORDER BY c1
 USING
 Mode (NONOVERLAPPING)
 Symbols (c2 = 'A' as A)
 Pattern ('A.A')
 Result
  (Accumulate (c2 OF A) AS x_pattern)
) AS dt;
```

Input

	c1	c2
1	1	A
2	2	A
3	3	A
4	4	A

Output

	x_pattern
1	[a, a]
2	[a, a]

After you have a complete Pattern match (TRUE), assuming there are more rows in the input table:

If **NONOVERLAPPING**, start next Pattern match on next row after last matched row

Lab 4c: Mode(Overlapping)

PATTERN(A.A) = Search for A followed by A

In **OVERLAPPING** match mode, **nPath** finds every occurrence of the pattern, regardless of whether it might have been part of a previously found match. This means that, in **OVERLAPPING** mode, a row can match more than once in that **PATTERN**

```
SELECT * FROM nPath
(ON matchup
 PARTITION BY c2
 ORDER BY c1
 USING
 Mode (OVERLAPPING)
 Symbols (c2 = 'A' as A)
 Pattern ('A.A')
 Result
 (Accumulate(c2 OF A) AS x_pattern)
) AS dt;
```

Input

	c1	c2
1	1	A
2	2	A
3	3	A
4	4	A

Output

	x_pattern
1	[A, A]
2	[A, A]
3	[A, A]

After you have a complete Pattern match (TRUE), assuming there are more rows in the input table:

If **OVERLAPPING**, go back to first row of last matched sequence, increment by one row and start here for your next Pattern match

Lab 4d: Mode Example (Which is correct answer?)

Query: Find which Products are Viewed by users ('view_product'), but not immediately Checked out ('checkout') in the following row (and count how many times that occurred)

```
SELECT product, count(*) as freq FROM nPath
(ON clicks
 PARTITION BY user_id, session_id
 ORDER BY datestamp
 USING
 SYMBOLS(page = 'view_product' AS P,
         page <> 'checkout' AS notC)
 PATTERN('P.notC')
 MODE( NONOVERLAPPING | OVERLAPPING )
 RESULT(first(product_id of P) as product)
)
GROUP by product ORDER by freq desc;
```

For USER_ID 84859:, which is 'right' answer?
Nonoverlap 1001-1, If Overlap , 1001 -1 1005 -1

	user_id	session_id	product_id	page	datestamp
1	84859	0	1001	view_product	2009-09-17 19:59:29
2	84859	0	1001	checkout	2009-09-17 19:59:59
3	84859	2	1001	view_product	2009-09-18 14:10:57
4	84859	2	1005	view_product	2009-09-18 14:14:59
5	84859	2	0	search	2009-09-18 14:18:11
6	84859	2	0	search_results	2009-09-18 14:19:28
7	84859	2	1005	search_results	2009-09-18 14:23:48
8	84859	2	1005	search_results	2009-09-18 14:26:56
9	84859	2	1005	search_results	2009-09-18 14:31:32

Output-Nonoverlapping

Output-Overlapping

	product	freq
1	1005	13789
2	1001	13643
3	1002	13549
4	1004	13539
5	1003	13301

Current Topic – Pattern with Symbols

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
- Symbols
- Mode
- **Pattern with Symbols**
- Result
- Daisy Chaining, Multiple Input, & Filter Argument
- Vantage Analyst
- Hackathon & Review

Pattern Operators

Use with pattern symbols to customize pattern-matching rules:

- '**.**' followed by (Use to separate a series of pattern symbols)
- '**|**' alternative (The equivalent of an OR)
- '**?**' occurs at most once (0-1)
- '*****' occurs zero or more times (0-n)
- '**+**' occurs at least once (1-n)
- '**^**' pattern must begin with value specified (also, value specified must be the first row within the partition)
- '**$**' pattern must end with

Customizing Pattern matching rules:

(X){a}	exactly A number of occurrences of X	**pattern('A.B{3}')**
(X){a,}	at least A number of occurrences of X	**pattern('A.B{2,}')**
(X){a,b}	A to B occurrences of X	**pattern('A.B{1,3}')**

Pattern Operators (cont.)

Operator	Description	Precedence
A	Matches one row that meets the definition of A	1 (highest)
A.	Matches one row that meets the definition of A	1
A?	Matches **0 or 1 rows** that satisfy the definition of A	1
A*	Matches **0 or more rows** that satisfy the definition of A (greedy operator)	1
A+	Matches **1 of more rows** that satisfy the definition of A (greedy operator)	1
A.B	Matches two rows, where the first row meets the definition of A and the second row meets the definition of B	2
A|B	Matches one row that meets the definition of either A or B	3

The **nPath** function uses 'GREEDY' pattern matching.
That is, it finds the longest available match when matching patterns.

Lab 5: Walking the Rows on Pattern Given Mode

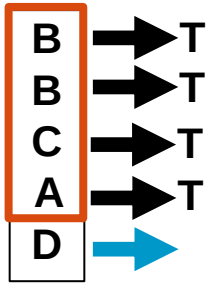

```
SELECT * FROM nPath
(ON npathBetween2
 PARTITION BY c2 ORDER BY c1
 USING
 Mode (NONOVERLAPPING)
 Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
 Pattern ('B+.C.A')
 Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

```
SELECT * FROM nPath
(ON npathBetween2
 PARTITION BY c2 ORDER BY c1
 USING
 Mode (OVERLAPPING)
 Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
 Pattern ('B+.C.A')
 Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

Input

	c1	c2	c3
1	1	1	B
2	2	1	B
3	3	1	C
4	4	1	A
5	5	1	D

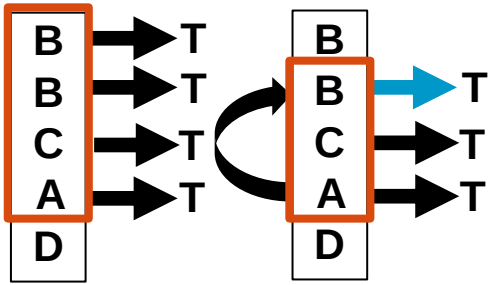
Match 1 of 1



With Nonoverlapping, after the first match, we start the next pattern on the **fifth** row

matches
[B, B, C, A]

Match 1 of 2 | Match 2 of 2



matches
[B, C, A]
[B, B, C, A]

With Overlapping, after the 1st match, start the next pattern on the **second** row of last match sequence

Lab 6: Followed By (.)

Here, we are searching for a ***B*** *followed by* ***C*** *followed by* ***A***

```
SELECT * FROM nPath
(ON npathBetween2
 PARTITION BY c2
 ORDER BY c1
 USING
 Mode (NONOVERLAPPING)
 Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
 Pattern ('B.C.A')
 Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

Input

	c1	c2	c3
1	1	1	B
2	2	1	B
3	3	1	C
4	4	1	A
5	5	1	D

Output

matches
[B, C, A]

Lab 7: OR (|)

Here, we are searching for a ***B, or C, or A***

```
SELECT * FROM nPath
(ON npathBetween2
 PARTITION BY c2
 ORDER BY c1
 USING
 Mode (NONOVERLAPPING)
 Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
 Pattern ('B|C|A')
 Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

Input

	c1	c2	c3
1	1	1	B
2	2	1	B
3	3	1	C
4	4	1	A
5	5	1	D

Output

matches
[B]
[B]
[C]
[A]

Lab 8: Followed by Together with OR

Here, we are searching for a ***B*** *followed by* ***C,*** *or an* ***A***

```
SELECT * FROM nPath
(ON npathBetween2
 PARTITION BY c2
 ORDER BY c1
 USING
 Mode (NONOVERLAPPING)
 Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
 Pattern ('B.C|A')
 Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

Input

	c1	c2	c3
1	1	1	B
2	2	1	B
3	3	1	C
4	4	1	A
5	5	1	D

Output

matches
[B, C]
[A]

Lab 9a: Parentheses

Look for path B, followed by either C or A

```
SELECT * FROM nPath
(ON npathBetween2
 PARTITION BY c2
 ORDER BY c1
 USING
 Mode (NONOVERLAPPING)
 Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
 Pattern ('B.(C|A)')
 Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

Input

	c1	c2	c3
1	1	1	B
2	2	1	B
3	3	1	C
4	4	1	A
5	5	1	D

Output

matches
[B, C]

- Absent parentheses, FOLLOWED BY [.] takes precedence over OR [|]
- Be aware of orders of operation and how the presence or absence of parentheses may impact your answer-sets

Lab 9b: Without Parentheses/with Parenthesis

No Parentheses - Look for path A.B or C [. takes precedence over |]

```
SELECT * FROM nPath
(ON npathBetween
PARTITION BY c2
ORDER BY c1
USING
Mode (NONOVERLAPPING)
Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
Pattern ('A.B|C')
Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)) AS dt;
```

Input

	c1	c2	c3
1	1	1	A
2	2	1	B
3	3	1	C
4	4	1	A
5	5	1	B
6	6	1	A
7	7	1	C

Output

matches
[A, B]
[C]
[A, B]
[C]

Parentheses - Look for path A.B or A.C [( ) takes precedence]

```
SELECT * FROM nPath
(ON npathBetween
PARTITION BY c2
ORDER BY c1
USING
Mode (NONOVERLAPPING)
Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
Pattern ('A.(B|C)')
Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)) AS dt;
```

Alternative code (without Parens):
Pattern ('A.B|A.C')

Input

	c1	c2	c3
1	1	1	A
2	2	1	B
3	3	1	C
4	4	1	A
5	5	1	B
6	6	1	A
7	7	1	C

Output

matches
[A, B]
[A, B]
[A, C]

Using LIKE Dependent on if Have CaseSpecific Table

- Teradata tables can be defined to be CASESPECIFIC. If enabled or not, this can change answer sets when using 'like' in nPath

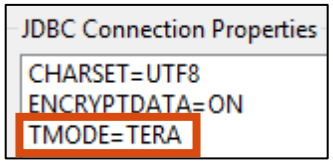
CaseSpecific table

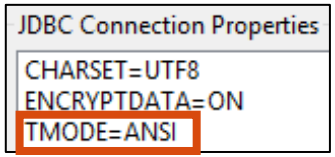
```
CREATE MULTISET TABLE jobs_CS
(emp INTEGER,
 job VARCHAR(20) CHARACTER SET LATIN CS,
 dt DATE FORMAT 'YYYY/MM/DD');
```

Non-CaseSpecific table

```
CREATE MULTISET TABLE jobs_nonCS
(emp INTEGER,
 job VARCHAR(20) CHARACTER SET LATIN,
 dt DATE FORMAT 'YYYY/MM/DD');
```

- If addition, TMode may affect 'like' queries:





1. If TMode=TD, queries are Case-insensitive (unless CREATE TABLE is CS) when using SQLE
2. If TMode=ANSI, then queries are Case-sensitive (unless CREATE TABLE is CS) when using SQLE

- Now that we know these concepts, let's do a few hands-on labs to confirm

Before We Proceed, Review of SYMBOL Predicates

We'll be using may of the below Predicates in the upcoming labs

Syntax

```
SYMBOLS ([column] operator [value] as Alias, … ) and/or
        TRUE as Alias (ie: any row)
```

Predicates	Description
TRUE AS <alias>	Symbol can match any row (often used for exploratory queries)
% _	% (any # of char) _ (positional) : Used for character comparisons
LIKE NOT LIKE	Case-insensitive (Teradata mode) or Case-sensitive (ANSI mode) text comparison on SQLE
LIKE NOT LIKE	Case-sensitive text comparison
ILIKE NOT ILIKE	Case-insensitive text comparison
LAG LEAD	Compare current row to prior/next row(s)
=, <, >, <=, >=, <>	Numeric comparisons

Lab 10a: LIKE on SQLE (with CS defined)

like (SQLE with TMode = TD / ANSI)

```
SELECT emp, jobs FROM nPath
(ON jobs_CS PARTITION BY emp ORDER BY dt
 USING
 Mode (NONOVERLAPPING)
 Symbols (job like 'soft%' as sw, TRUE as next_job)
 Pattern ('sw.next_job')
 Result  (First (emp of sw) as emp,
          Accumulate(job of any(sw, next_job)) as jobs)
) AS dt ORDER BY emp;
```

Input

	emp	job	dt
1	1	Software Eng	2010-01-01
2	1	Dir Eng	2010-02-01
3	1	CTO	2010-03-01
4	2	janitor	2010-01-01
5	2	soft dev	2010-02-01
6	2	cust_svc	2010-03-01
7	2	mgr	2010-04-01

Output

emp	jobs
2	[soft dev, cust_svc]

Lab 10b: LIKE on SQLE (with CS NOT defined)

like (SQLE with TMode = TD)

TD Studio only

```
SELECT emp, jobs FROM nPath
(ON jobs_notCS PARTITION BY emp ORDER BY dt
 USING
 Mode (NONOVERLAPPING)
 Symbols (job like 'soft%' as sw, TRUE as next_job)
 Pattern ('sw.next_job')
 Result (First (emp of sw) as emp,
         Accumulate(job of any(sw, next_job)) as jobs)
) AS dt ORDER BY emp;
```

JDBC Connection Properties
CHARSET=UTF8
ENCRYPTDATA=ON
TMODE=TERA

Input

	emp	job	dt
1	1	Software Eng	2010-01-01
2	1	Dir Eng	2010-02-01
3	1	CTO	2010-03-01
4	2	janitor	2010-01-01
5	2	soft dev	2010-02-01
6	2	cust_svc	2010-03-01
7	2	mgr	2010-04-01

Output

emp	jobs
1	[Software Eng, Dir Eng]
2	[soft dev, cust_svc]

like (SQLE with TMode = ANSI)

TD Studio only

```
SELECT emp, jobs FROM nPath
(ON jobs_notCS PARTITION BY emp ORDER BY dt
 USING
 Mode (NONOVERLAPPING)
 Symbols (job like 'soft%' as sw, TRUE as next_job)
 Pattern ('sw.next_job')
 Result (First (emp of sw) as emp,
         Accumulate(job of any(sw, next_job)) as jobs)
) AS dt ORDER BY emp;
```

JDBC Connection Properties
CHARSET=UTF8
ENCRYPTDATA=ON
TMODE=ANSI

Output

emp	jobs
2	[soft dev, cust_svc]

Editor client defaults to TMode = TERA

Lab 11a: Exploring the ^ Predicate with LIKE

- Recall that the ^ predicate forces the **Pattern** to begin with whatever you specify. Furthermore, if it is the first alias listed in PATTERN, it forces the value specified to be the first row of the partition
- This lab will focus on the Input data displayed below
- We will show an example of each of the following for our **Pattern**:
  - **Pattern ('sw.next_job')**
  - **Pattern ('^sw.next_job')**

```
SELECT *
FROM jobs2
ORDER BY emp, dt;
```

Input

	emp	job	dt
1	1	Software Eng	2010-01-01
2	1	Dir Eng	2010-02-01
3	1	CTO	2010-03-01
4	2	janitor	2010-01-01
5	2	soft dev	2010-02-01
6	2	cust_svc	2010-03-01
7	2	mgr	2010-04-01

Lab 11b: Without ^ Predicate and with LIKE

Query: Accumulate any job per partition ilike '%soft%', followed by next job

```
SELECT emp, jobs
FROM nPath
(ON jobs2
 PARTITION BY emp
 ORDER BY dt
 USING
 Mode (NONOVERLAPPING)
 Symbols (Lower(job) LIKE '%soft%' as
sw, TRUE as next_job)
 Pattern ('sw.next_job')
 Result (First(emp of sw) as emp,
         Accumulate(job of
         any(sw, next_job)) as jobs)
) AS dt
ORDER BY emp;
```

TRUE as alias = Next row

- Here, we are not specifying the ^ predicate in our **Pattern** argument
- So, we're searching for any job-path pattern that goes FROM '%soft%' to anything else
- Both **emp** values have met our conditions

Input

	emp	job	dt
1	1	Software Eng	2010-01-01
2	1	Dir Eng	2010-02-01
3	1	CTO	2010-03-01
4	2	janitor	2010-01-01
5	2	soft dev	2010-02-01
6	2	cust_svc	2010-03-01
7	2	mgr	2010-04-01

Output

emp	jobs
1	[Software Eng, Dir Eng]
2	[soft dev, cust_svc]

Lab 11c: With ^ Predicate and with LIKE

Query: Accumulate if 1st job in partition like '%soft%', followed by next job

```
SELECT emp, jobs
FROM nPath
(ON jobs2
 PARTITION BY emp
 ORDER BY dt
 USING
 Mode (NONOVERLAPPING)
 Symbols (Lower(job) LIKE '%soft%'
as sw, TRUE as next_job)
 Pattern ('^sw.next_job')
 Result (First(emp of sw) as emp,
         Accumulate(job of
         any(sw,next_job)) as jobs)
) AS dt
ORDER BY emp;
```

- Here, we are specifying the ^ predicate in our **Pattern** argument
- Given this, we are searching for any job-path pattern that *begins the partition* with '%soft%', and then goes to anything else, whatever it may be
- Only **emp = 1** has the first row of its partition beginning with '%soft%', thus the job path of **emp 1** is returned, but not the job path of **emp = 2** (which begins with 'Janitor')

Input

	emp	job	dt
1	1	Software Eng	2010-01-01
2	1	Dir Eng	2010-02-01
3	1	CTO	2010-03-01
4	2	janitor	2010-01-01
5	2	soft dev	2010-02-01
6	2	cust_svc	2010-03-01
7	2	mgr	2010-04-01

Output

emp	jobs
1	[Software Eng, Dir Eng]

Lab 11d: With ^ Predicate and with NOT LIKE

Query: Accumulate 1st job in partition that did not start ilike '%soft%', followed by next job

```
SELECT emp, jobs
FROM nPath
(ON jobs2
 PARTITION BY emp
 ORDER BY dt
 USING
 Mode (NONOVERLAPPING)
 Symbols (job NOT LIKE '%soft%' AND job NOT LIKE '%Soft%'
as notsw, TRUE as next_job)
 Pattern ('^notsw.next_job')
 Result (First(emp of notsw) as emp,
         Accumulate(job of any(notsw, next_job)) as jobs)
) AS dt
ORDER BY emp;
```

Input

	emp	job	dt
1	1	Software Eng	2010-01-01
2	1	Dir Eng	2010-02-01
3	1	CTO	2010-03-01
4	2	janitor	2010-01-01
5	2	soft dev	2010-02-01
7	2	mgr	2010-04-01

Output

emp	jobs
2	[janitor, soft dev]

Lab 12a: Using the + Predicate

- Recall the plus symbol (**+**) signifies *occurs at least once (1-n)*
- In the examples below, for each **emp_id** after a match on **sw**, the **Pattern ('sw.next_job')** will return only the very next job, whereas the **Pattern ('sw.next_job+')** will return all subsequent jobs

Input

emp_id	job_desc	bgn_dt	end_dt
1	Software Engineer	2010-01-01	2010-12-31
1	Director of Engineering	2011-01-01	2011-12-31
1	CTO	2012-01-01	2012-12-31
2	Janitor	2013-01-01	2013-12-31
2	Software Engineer	2014-01-01	2014-12-31
2	Director of Engineering	2015-01-01	2015-12-31
2	CTO	2016-01-01	2016-12-31

Without + (Display 1st job that has like **'%soft%'**, followed by next job/partition)

```
SELECT emp_id, Matches, count (*) FROM nPath  (ON
jobs PARTITION BY emp_id ORDER BY bgn_dt USING Mode
(NONOVERLAPPING)
Symbols (Lower(job_desc) like 'soft%' as sw,
         TRUE as next_job)
Pattern ('sw.next_job')
Result (First(emp_id of sw) as emp_id,
Accumulate(job_desc of any(sw,next_job)) as Matches
) AS dt GROUP BY emp_id, Matches ORDER BY emp_id;
```

emp_id	matches	Count(*)
1	[Software Engineer, Director of Engineering]	1
2	[Software Engineer, Director of Engineering]	1

With + (Display 1st job that has like **'%soft%'**, followed by 1-more jobs/partition)

```
SELECT emp_id, Matches, count (*) FROM nPath  (ON
jobs PARTITION BY emp_id ORDER BY bgn_dt USING Mode
(NONOVERLAPPING)
Symbols (Lower(job_desc) like 'soft%' as sw,
         TRUE as next_job)
Pattern ('sw.next_job+')
Result (First(emp_id of sw) as emp_id,
Accumulate(job_desc of any(sw,next_job)) as Matches
) AS dt GROUP BY emp_id, Matches ORDER BY emp_id;
```

emp_id	matches	Count(*)
1	[Software Engineer, Director of Engineering, CTO]	1
2	[Software Engineer, Director of Engineering, CTO]	1

Lab 12b: Comparing + (1-n) to * (0-n) Predicate

- Recall the plus symbol (**+**) signifies *occurs at least once (1-n)* whereas asterisk * symbol (*) signifies *0 to infinity (0-n)*

Input

	emp_id	job_desc	bgn_dt	end_dt
1	1	Software Engineer	2010-01-01	2010-12-31
2	1	Director of Engineering	2011-01-01	2011-12-31
3	1	CTO	2012-01-01	2012-12-31
4	2	Janitor	2013-01-01	2013-12-31
5	2	software dev	2014-01-01	2014-12-31
6	2	Director of Engineering	2015-01-01	2015-12-31
7	2	CTO	2016-01-01	2016-12-31
8	3	software tester	2010-01-01	

This person only has 1 row in his Partition

With + (Display 1st job that has like **'%soft%'**, followed by 1- more jobs/partition**)**

```
SELECT emp_id, Matches, count (*) FROM nPath  (ON jobs3
PARTITION BY emp_id ORDER BY bgn_dt USING Mode (NONOVERLAPPING)
Symbols (Lower(job_desc) like 'soft%' as sw, TRUE as next_job)
Pattern ('sw.next_job+')
Result (First(emp_id of sw) as emp_id,
Accumulate(job_desc of any(sw,next_job)) as Matches
) AS dt GROUP BY emp_id, Matches ORDER BY emp_id;
```

emp_id	matches	Count(*)
1	[Software Engineer, Director of Engineering, CTO]	1
2	[Software Engineer, Director of Engineering, CTO]	1

Lab 12c: ^ (Starts with) and $ (Ends with) Predicate

- Recall the ^ symbol signifies *starts with* whereas the $ symbol signifies *ends with*

Input

	emp_id	job_desc	bgn_dt	end_dt
1	1	Software Engineer	2010-01-01	2010-12-31
2	1	Director of Eng	2011-01-01	2011-12-31
3	1	CTO	2012-01-01	2012-12-31
4	2	Janitor	2013-01-01	2013-12-31
5	2	software dev	2014-01-01	2014-12-31
6	2	Mkt Mgr	2015-01-01	2015-12-31
7	3	software tester	2010-01-01	2015-12-31
8	3	Mkt Mgr	2016-01-01	2016-12-31
9	4	junior_mkt	2010-01-01	

With $ (Find all 3-row Partitions)

```
SELECT * FROM nPath
(ON jobs4
 PARTITION BY emp_id ORDER BY bgn_dt
 USING
 Mode (NONOVERLAPPING)
 Symbols (TRUE as next_row)
 Pattern ('next_row{2}.next_row$')
 Result (First(emp_id of any(next_row)) as emp_id,
         Accumulate(job_desc of any(next_row)) as
Matches)
) AS dt ORDER BY emp_id;
```

Alt Pattern: 'next_row{3}$'

emp_id	matches
1	[Software Engineer, Director of Eng, CTO]
2	[Janitor, software dev, Mkt Mgr]

With ^, $ (Find 2-row Partitions who 1st row starts with job = 'soft%' and ends with any row

```
SELECT * FROM nPath
(ON jobs4
 PARTITION BY emp_id  ORDER BY bgn_dt
 USING
 Mode (NONOVERLAPPING)
 Symbols (TRUE as next_row, Lower(job_desc) like
'soft%'as soft) Pattern ('^soft.next_row$')
  Result (First(emp_id of any(soft, next_row)) as
  emp_id, Accumulate(job_desc of any(soft,
  next_row)) as Matches)) AS dt ORDER BY emp_id;
```

emp_id	matches
3	[software tester, Mkt Mgr]

Lab 13a: True Predicate – View the Data

- Here, we are familiarizing ourselves with the **bank_web_clicks** table
- On the following page, we will use the **TRUE** predicate to discover which web paths are the most-commonly travelled

```
SELECT * FROM bank_web_clicks
WHERE customer_id IN (11603)
ORDER BY customer_id, datestamp;
```

```
CREATE MULTISET TABLE bank_web_clicks ,FALLBACK ,
     NO BEFORE JOURNAL,
     NO AFTER JOURNAL,
     CHECKSUM = DEFAULT,
     DEFAULT MERGEBLOCKRATIO,
     MAP = TD_MAP1
     (
      customer_id INTEGER,
      session_id INTEGER,
      page VARCHAR(100) CHARACTER SET LATIN NOT CASESPECIFIC,
      datestamp TIMESTAMP(6))
PRIMARY INDEX ( customer_id );
```

	customer_id	session_id	page	datestamp
1	11603	0	ACCOUNT SUMMARY	2004-04-25 08:37:17.000000
2	11603	0	FUNDS TRANSFER	2004-04-25 08:39:50.000000
3	11603	0	CUSTOMER SUPPORT	2004-04-25 08:41:21.000000
4	11603	0	ACCOUNT SUMMARY	2004-04-25 08:42:45.000000
5	11603	0	ACCOUNT SUMMARY	2004-04-25 08:43:16.000000
6	11603	0	FUNDS TRANSFER	2004-04-25 08:45:21.000000
7	11603	0	ACCOUNT HISTORY	2004-04-25 08:49:00.000000
8	11603	0	FAQ	2004-04-25 08:51:40.000000
9	11603	0	FUNDS TRANSFER	2004-04-25 08:55:24.000000
10	11603	0	ACCOUNT SUMMARY	2004-04-25 08:56:18.000000

Lab 13b: True Predicate

Query: Accumulate all web pages. Also, count the # of times these paths occurred

'true as <alias>' = 'next row' in partition

```
SELECT path, count(*) AS occurs
FROM nPath
(ON bank_web_clicks
 PARTITION BY customer_id, session_id
 ORDER BY datestamp
 USING
 Mode (NONOVERLAPPING)
 Symbols (TRUE AS wPAGE)
 Pattern ('wPAGE+')
 Result (
 Accumulate(page OF ANY (wPAGE)) as path)
) AS dt
GROUP BY path
HAVING occurs >= 500
ORDER BY occurs DESC;
```

Output

	path	occurs
1	[ACCOUNT SUMMARY, FAQ]	2854
2	[ACCOUNT SUMMARY, VIEW DEPOSIT DETAILS]	2830
3	[ACCOUNT SUMMARY, FUNDS TRANSFER]	2746
4	[ACCOUNT SUMMARY, ACCOUNT SUMMARY]	2715
5	[ACCOUNT SUMMARY, ACCOUNT HISTORY]	2699
6	[ACCOUNT SUMMARY, ONLINE STATEMENT ENROLLMENT]	698
7	[ACCOUNT SUMMARY, PROFILE UPDATE]	684
8	[ACCOUNT SUMMARY, CUSTOMER SUPPORT]	647
9	[ACCOUNT SUMMARY, VIEW DEPOSIT DETAILS, ACCOUNT HISTORY]	575
10	[ACCOUNT SUMMARY, FUNDS TRANSFER, ACCOUNT SUMMARY]	571
11	[ACCOUNT SUMMARY, ACCOUNT HISTORY, ACCOUNT SUMMARY]	549
12	[ACCOUNT SUMMARY, ACCOUNT SUMMARY, FAQ]	545
13	[ACCOUNT SUMMARY, VIEW DEPOSIT DETAILS, FUNDS TRANSFER]	524
14	[ACCOUNT SUMMARY, ACCOUNT SUMMARY, ACCOUNT HISTORY]	510
15	[ACCOUNT SUMMARY, FUNDS TRANSFER, VIEW DEPOSIT DETAILS]	507
16	[ACCOUNT SUMMARY, FUNDS TRANSFER, ACCOUNT HISTORY]	502

Lab 14: LAG in SYMBOLS

Query: Find customers who view a product (P), then within 15 seconds buy it in next click

```
SELECT * from nPath
(ON pageview
 PARTITION BY userid
 ORDER BY ts
 USING
 MODE(nonoverlapping)
 SYMBOLS(url like'p%.html' as P,
         url='checkout.html' and
         LAG(ts,1) >= ts - interval '15' second as BUY)
 PATTERN('P.BUY')
 RESULT(FIRST(userid of P) as userid,
        FIRST(ts of P) as p_ts,
        FIRST(ts of BUY) as buy_ts)) as dt;
```

Input

	userid	ts	url
1	mike	2011-01-01 04:50:10.000000	home.html
2	mike	2011-01-01 04:53:23.000000	prod1.html
3	mike	2011-01-01 04:55:20.000000	prod2.html
4	mike	2011-01-01 04:55:30.000000	checkout.html
5	mike	2011-01-01 05:00:05.000000	logout.html

Output

	userid	p_ts	buy_ts
1	mike	2011-01-01 04:55:20.000000	2011-01-01 04:55:30.000000

Lab 15: LEAD in SYMBOLS

Query: After taking a Quiz, how many students clicked on SCORE hotlink within 10 seconds?

```
SELECT * from nPath
(ON quizme
 PARTITION BY userid ORDER BY ts
 USING
 MODE(nonoverlapping)
 SYMBOLS(url like 'quiz%' and
         LEAD(ts,1) <= ts + interval '10' second and
         LEAD(url,1) = 'score' as Q,
         url = 'score' as S)
 PATTERN('Q.S')
 RESULT(FIRST(userid of Q) as userid,
         FIRST(ts of Q) as q_ts,
         LAST(ts of S) as s_ts));
```

Input

	userid	score	url	ts
1	1		quiz1	2015-04-20 10:00:00.000000
2	1	75	score	2015-04-20 10:00:10.000000
3	2		faq	2015-04-20 10:00:00.000000
4	2		cust_svc	2015-04-20 10:00:10.000000
5	3		quiz2	2015-04-20 10:00:00.000000
6	3	50	score	2015-04-20 10:00:30.000000

Output

	userid	q_ts	s_ts
1	1	2015-04-20 10:00:00.000000	2015-04-20 10:00:10.000000

Current Topic – Result

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
- Symbols
- Mode
- Pattern with Symbols
- **Result**
- Daisy Chaining, Multiple Input, & Filter Argument
- Vantage Analyst
- Hackathon & Review

Result – What You Wish to Project in Answer Set

- **RESULT** outputs each matched **PATTERN** (along with SELECT clause and any aggregates)
- **nPath** generates a row of output that can contain SQL and/or ML aggregates computed over the rows within the matched **PATTERN**

Syntax

```
Result ({ aggregate_function (col_expr OF symbol) AS alias_1 }[,...])
```

```
SELECT emp_id, Matches, count (*)
FROM nPath
(ON jobs PARTITION BY emp_id ORDER BY bgn_dt
 USING
 Mode (NONOVERLAPPING)
 Symbols (Lower(job_desc) like '%soft%' AS sw,
 TRUE AS next_job)
 Pattern ('sw.next_job')
 Result (First(emp_id OF sw) AS emp_id,
         Accumulate(job_desc OF ANY(sw,next_job)) AS Matches)
) AS dt
GROUP BY emp_id, Matches
ORDER BY emp_id;
```

- Here, we are using **First** and **Accumulate** in our **Result** argument
- **First** returns the *col_expr* value of the first matched row
- **Accumulate** returns, for each matched row, the concatenated values in *col_expr*, separated by a delimiter. The default delimiter is a comma followed by a blank space (,)

Result Aggregations

Following are some common Aggregates that you can specify in the Result argument:

- **COUNT**(* of SYMBOL list)
- **FIRST**(<col_expr> of SYMBOL list)
- **LAST**(<col_expr> of SYMBOL list)
- **NTH**(<col_expr>, n of SYMBOL list)
- **SUM**(<col_expr> of SYMBOL list)
- **AVG**(<col_expr> of SYMBOL list)
- **MAX**(<col_expr> of SYMBOL list)
- **MIN**(<col_expr> of SYMBOL list)
- **ACCUMULATE**(<expression> of SYMBOL list)
- **ANY**: e.g., SUM(<expression> of ANY(A,B,C))

- **FIRST_NOTNULL** (column of SYMBOL list): Returns first non-null row that maps to SYMBOL list
- **LAST_NOTNULL** (column of SYMBOL list): Returns last non-null row that maps to SYMBOL list
- **MAX_CHOOSE** (qty column, column name): Returns descriptive column of highest qty col
- **MIN_CHOOSE** (qty column, column name): Returns descriptive column of lowest qty col
- **DUPCOUNT**: Counts # of times value has appeared preceding this row
- **DUPCOUNTCUM**: # of Duplicate values have appeared contiguously preceding

Lab 16a: Result Aggregate Example

Query: Find all Products purchased by Customers along with Sales Quantity aggregates

```
SELECT * FROM nPath
(ON sales_fact PARTITION BY customer_id ORDER BY sales_date
 USING
 MODE(NONOVERLAPPING)
 SYMBOLS (TRUE as A)
 PATTERN ('A*')
 RESULT (FIRST(customer_id of A) as cust,       COUNT(* of A) as cnt,
         SUM(sales_quantity of A) as sum_prod, AVG(sales_quantity of A) as avg_prod,
         MIN(sales_quantity of A) as min_prod, MAX(sales_quantity of A) as max_prod,
        FIRST(product_id of A) as first_prod, LAST(product_id of A) as last_prod,
  NTH(product_id,2 of A) as nth_prod,ACCUMULATE(product_id of ANY(A)) as all_prods)
);
```

cust	cnt	sum_qty	avg_qty	min_qty	max_qty	first_prod	last_prod	second_prod	all_prods
1	27	138	5.11111…	1	10	74	93	57	[74, 57, 30, 46, 48, 64, 11, 81, 81, 80, 24, 66, 83, 87, 15, 20, 75, 59, 50, 36, 95, 42, 11, 5, 55, 97, 93]
9	310	1587	5.1193…	1	10	76	67	69	[76, 69, 97, 70, 70, 63, 73, 5, 56, 63, 6, 9, 5, 100, 27, 49, 92, 44, 96, 8, 92, 56, 30, 66, 28, 92, 81, 27,
2	22	127	5.7727…	1	10	76	19	49	[76, 49, 100, 28, 94, 41, 43, 5, 18, 83, 28, 35, 49, 11, 53, 64, 55, 100, 91, 99, 55, 19]
10	92	479	5.2065…	1	10	38	43	43	[38, 43, 31, 61, 60, 19, 64, 87, 53, 61, 20, 2, 75, 74, 16, 70, 54, 51, 66, 34, 89, 37, 47, 53, 67, 81, 82,
3	105	584	5.5619…	1	10	6	24	34	[6, 34, 94, 5, 52, 59, 24, 29, 85, 94, 54, 7, 87, 38, 76, 56, 66, 44, 27, 27, 84, 92, 8, 59, 48, 92, 9, 79, 9
11	102	546	5.3529…	1	10	73	33	31	[73, 31, 54, 44, 42, 85, 10, 22, 75, 90, 66, 62, 36, 60, 96, 88, 96, 39, 17, 10, 46, 29, 90, 11, 49, 51, 59,
4	5	29	5.8	2	10	45	13	21	[45, 21, 34, 80, 13]

Lab 16b: Result Nth Aggregate (Sales Velocity)

Query: Find all Products purchased by Customers and the Sales Velocity between visits

```
SELECT cust, sales1, sales2, sales3,
       sales2 - sales1 as sales_velocity_2nd_visit,
       sales3 - sales2 as sales_velocity_3rd_visit
 FROM nPath
(ON sales_fact
 PARTITION BY customer_id
 ORDER BY sales_date
 USING
 MODE(NONOVERLAPPING)
 SYMBOLS (TRUE as A)
 PATTERN ('A*')
 RESULT (FIRST(customer_id of A) as cust,
         COUNT(* of A) as cnt,
         NTH(sales_quantity, 1 of A) as sales1,
         NTH(sales_quantity, 2 of A) as sales2,
         NTH(sales_quantity, 3 of A) as sales3)
);
```

Output

	cust	sales1	sales2	sales3	sales_velocity_2nd_visit	sales_velocity_3rd_visit
1	1	9	5	7	-4	2
2	2	3	8	9	5	1
3	3	5	2	8	-3	6
4	4	2	10	10	8	0
5	5	7	9	7	2	-2
6	6	4	9	7	5	-2
7	7	9	2	8	-7	6

Lab 16c: View the Data

- Over the next many pages, we will walk through various **Result** arguments using the **web_purchases** table, shown below
- Note:
  - There are three **user_id** values
  - **user_id 1** had two sessions. The other two **user_id** values had only one **session_id** each
  - **user_id 1** bought a *guitar*, **user_id 2** bought *strings* as well as a *guitar*, and **user_id 3** bought a *capo*

```
SELECT * FROM web_purchases
ORDER BY user_id, date_time;
```

	user_id	session_id	date_time	page	product	prod_qty	prod_price
1	1	0	2018-01-01 13:21:01.000000	home			
2	1	0	2018-01-01 13:22:01.000000	contact_us			
3	1	0	2018-01-01 13:23:01.000000	view_prod	guitar	1	
4	1	0	2018-01-01 13:24:01.000000	checkout	guitar	1	250.00
5	1	1	2018-01-02 14:21:01.000000	home			
6	1	1	2018-01-02 14:22:01.000000	about_us			
7	2	0	2018-02-01 13:21:01.000000	home			
8	2	0	2018-02-01 13:22:01.000000	view_prod	strings		
9	2	0	2018-02-01 13:23:01.000000	view_prod	guitar		
10	2	0	2018-02-01 13:24:01.000000	checkout	strings	5	25.00
11	2	0	2018-02-01 14:25:01.000000	checkout	guitar	1	1000.00
12	3	0	2018-04-01 14:22:01.000000	home			
13	3	0	2018-04-01 14:23:01.000000	view_prod	capo		
14	3	0	2018-04-01 14:24:01.000000	checkout	capo	1	15.00

Lab 16d: Result Argument: Accumulate

Query: Accumulate Products purchased of 1-m 'view_prod' page followed by 1-m 'checkout' page

```
SELECT * FROM nPath
(ON web_purchases
 PARTITION BY user_id, session_id ORDER BY date_time
 USING
 Symbols(page = 'view_prod' as V, page = 'checkout' as C)
 Mode(NONOVERLAPPING) Pattern ('V+.C+')
 Result(Accumulate(product of any(V, C)) as Matches)) AS dt;
```

- In the example, we are searching within each partition for a **Pattern** of *one or more instances of* ***view_prod****, followed by one or more instances of* ***checkout***
- Our **Accumulate** argument returns any instances of product that meet the conditions of our **Pattern** within each partition

Input

	user_id	session_id	date_time	page	product	prod_qty	prod_price
1	1	0	2018-01-01 13:21:01.000000	home			
2	1	0	2018-01-01 13:22:01.000000	contact_us			
3	1	0	2018-01-01 13:23:01.000000	view_prod	guitar	1	
4	1	0	2018-01-01 13:24:01.000000	checkout	guitar	1	250.00
5	1	1	2018-01-02 14:21:01.000000	home			
6	1	1	2018-01-02 14:22:01.000000	about_us			
7	2	0	2018-02-01 13:21:01.000000	home			
8	2	0	2018-02-01 13:22:01.000000	view_prod	strings		
9	2	0	2018-02-01 13:23:01.000000	view_prod	guitar		
10	2	0	2018-02-01 13:24:01.000000	checkout	strings	5	25.00
11	2	0	2018-02-01 14:25:01.000000	checkout	guitar	1	1000.00
12	3	0	2018-04-01 14:22:01.000000	home			
13	3	0	2018-04-01 14:23:01.000000	view_prod	capo		
14	3	0	2018-04-01 14:24:01.000000	checkout	capo	1	15.00

Output

	matches
1	[guitar, guitar]
2	[strings, guitar, strings, guitar]
3	[capo, capo]

Lab 16e: Result Arguments: First, Accumulate


```
SELECT * FROM nPath
(ON web_purchases
PARTITION BY user_id, session_id ORDER BY date_time
USING Mode (NONOVERLAPPING)
Symbols (page = 'view_prod' as V, page='checkout' as C)
Pattern ('V+.C+')
Result (First (user_id of V) as user_id,
        First (session_id of V) as session_id,
        Accumulate (product of any(V, C)) as Matches)
) AS dt ORDER BY user_id, session_id;
```

- Our **First** arguments return the very first instance of **user_id** that viewed the specified product within the partition
- Our **Accumulate** argument returns any instances of product that meet the conditions of our **Pattern** within the partition

Input

	user_id	session_id	date_time	page	product	prod_qty	prod_price
1	1	0	2018-01-01 13:21:01.000000	home			
2	1	0	2018-01-01 13:22:01.000000	contact_us			
3	1	0	2018-01-01 13:23:01.000000	view_prod	guitar	1	
4	1	0	2018-01-01 13:24:01.000000	checkout	guitar	1	250.00
5	1	1	2018-01-02 14:21:01.000000	home			
6	1	1	2018-01-02 14:22:01.000000	about_us			
7	2	0	2018-02-01 13:21:01.000000	home			
8	2	0	2018-02-01 13:22:01.000000	view_prod	strings		
9	2	0	2018-02-01 13:23:01.000000	view_prod	guitar		
10	2	0	2018-02-01 13:24:01.000000	checkout	strings	5	25.00
11	2	0	2018-02-01 14:25:01.000000	checkout	guitar	1	1000.00
12	3	0	2018-04-01 14:22:01.000000	home			
13	3	0	2018-04-01 14:23:01.000000	view_prod	capo		
14	3	0	2018-04-01 14:24:01.000000	checkout	capo	1	15.00

Output

	user_id	session_id	matches
1	1	0	[guitar, guitar]
2	2	0	[strings, guitar, strings, guitar]
3	3	0	[capo, capo]

Lab 16f: Result Arguments: First, Accumulate, Count, Sum


```
SELECT * FROM nPath
(ON web_purchases
 PARTITION BY user_id, session_id
 ORDER BY date_time
 USING
 Mode(NONOVERLAPPING)
 Symbols(page = 'view_prod' AS V,
         page='checkout' AS C)
 Pattern('V+.C+')
 Result(First (user_id of V) AS user_id,
  First (session_id of V) AS session_id,
  Accumulate (product of any(V, C)) AS Matches,
  Count (distinct product of any(C)) AS cd_Matches,
  Sum (cast(prod_price as integer) of any (C))
        AS total_price)
) AS dt ORDER BY user_id, session_id;
```

Input

	user_id	session_id	date_time	page	product	prod_qty	prod_price
1	1	0	2018-01-01 13:21:01.000000	home			
2	1	0	2018-01-01 13:22:01.000000	contact_us			
3	1	0	2018-01-01 13:23:01.000000	view_prod	guitar	1	
4	1	0	2018-01-01 13:24:01.000000	checkout	guitar	1	250.00
5	1	1	2018-01-02 14:21:01.000000	home			
6	1	1	2018-01-02 14:22:01.000000	about_us			
7	2	0	2018-02-01 13:21:01.000000	home			
8	2	0	2018-02-01 13:22:01.000000	view_prod	strings		
9	2	0	2018-02-01 13:23:01.000000	view_prod	guitar		
10	2	0	2018-02-01 13:24:01.000000	checkout	strings	5	25.00
11	2	0	2018-02-01 14:25:01.000000	checkout	guitar	1	1000.00
12	3	0	2018-04-01 14:22:01.000000	home			
13	3	0	2018-04-01 14:23:01.000000	view_prod	capo		
14	3	0	2018-04-01 14:24:01.000000	checkout	capo	1	15.00

Output

	user_id	session_id	matches	cd_matches	total_price
1	1	0	[guitar, guitar]	1	250
2	2	0	[strings, guitar, strings, guitar]	2	1025
3	3	0	[capo, capo]	1	15

- Our **Sum** argument sums the **prod_price** of qualifying *checkout* products
- Our **Count** argument counts the distinct products that were present in a ***checkout***
- Our **First** arguments return the very first instance of **user_id** that viewed the specified product
- Our **Accumulate** argument returns any instances of product that meet the conditions of our **Pattern**

Current Topic – Daisy Chaining, Multiple Input and Filter

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
- Symbols
- Mode
- Pattern with Symbols
- Result
- **Daisy Chaining, Multiple Input, & Filter Argument**
- Vantage Analyst
- Hackathon & Review

Multiple-Input nPath Concepts

Multiple-Input nPath can jointly analyze multiple sequences of different event types

Benefits of multiple inputs for nPath

- **Eliminates Unions**: With single-input nPath you must merge multiple fact tables via union and sort the combined data set. With multiple inputs, individual tables are sorted and merged within nPath
- **Eliminates Joins**: For some applications, a join is required to merge a dimension table with a fact table into a single input. Multi-input nPath supports dimension inputs to avoid this
- **Eliminates Casting**: In single-input case you must merge data from multiple streams and may need to cast to a common data type

Lab 17a: nPath Multiple Input – Two Tables

Consider path analysis with two seemingly unrelated data sets:

- Multi_clicks table contains basic clickstream data which is already sessionized
- Multi_ads table contains all the information which was aired on a tv channel

Query: Find which channel advertisement had a sale, product(s) bought, and path taken to purchase

Multi_clicks

userid	sessionid	ts	page
67403	14	10-10-2011 13:18:30	home
67403	14	10-10-2011 13:18:31	product1
67403	14	10-10-2011 13:18:32	product2
67403	14	10-10-2011 13:18:40	checkout
67403	14	10-10-2011 13:19:00	product3

Multi_ads

Adid	channel	ts
10	msnbc	10-10-2011 07:00:20
11	espn	10-10-2011 13:18:00
12	nbc	10-10-2011 15:34:26
10	msnbc	10-10-2011 15:35:00

Lab 17b: Multiple Input nPath

The multi_clicks table has a PARTITION BY clause while the multi_ads table becomes a DIMENSION table. Both are ordered by **ts** column which sorts them prior to processing.

```
SELECT * FROM nPath
(ON TRNG_TDU_TD01.multi_clicks
 PARTITION BY userid, sessionid ORDER BY ts
 ON TRNG_TDU_TD01.multi_ads DIMENSION ORDER BY ts
 USING
 MODE(nonoverlapping)
 SYMBOLS(multi_ads.adid IS NOT NULL as A,
         multi_clicks.page <> 'checkout' as X,
         multi_clicks.page = 'checkout' as C)
 PATTERN('A.X+.C')
 RESULT(first(multi_ads.channel of A) as Channel,
      accumulate(multi_clicks.page of X) as ClickPath,
      first(multi_clicks.sessionid of C) as SessionID));
```

Adid	channel	ts
10	msnbc	10-10-2011 07:00:20
11	espn	10-10-2011 13:18:00

userid	sessionid	ts	page
67403	14	10-10-2011 13:18:30	home
67403	14	10-10-2011 13:18:31	product1
67403	14	10-10-2011 13:18:32	product2
67403	14	10-10-2011 13:18:40	checkout

Adid	channel	ts
12	nbc	10-10-2011 15:34:26
10	msnbc	10-10-2011 15:35:00

Based on Output, we can infer that the ESPN advertisement leads to sales of Product1 and Product2

Output

	channel	clickpath	sessionid
1	espn	[home, product1, product2]	14

Lab 18: nPath FILTER Argument

The (optional) FILTER argument lets you impose additional filters on the pattern matches being found by nPath. This clause lets you compare the current value of a row with a value from an unknown number of rows forward or behind (LAG/LEAD can't do this).

Input

	userid	sessionid	clickts	page
1	1	1	2019-02-27 10:15:00.000000	home
2	1	1	2019-02-27 10:16:00.000000	view
3	1	1	2019-02-27 10:17:00.000000	view
4	1	1	2019-02-27 10:18:00.000000	checkout
5	1	1	2019-02-27 10:19:00.000000	checkout
6	1	1	2019-02-27 10:20:00.000000	view
7	1	1	2019-03-10 10:21:00.000000	view
8	1	1	2019-03-10 10:22:00.000000	view
9	2	1	2019-03-10 11:34:00.000000	home
10	2	1	2019-03-10 11:41:00.000000	view
11	2	1	2019-03-10 11:45:00.000000	view
12	2	1	2019-03-10 11:52:00.000000	checkout
13	2	1	2019-03-10 11:58:00.000000	view
14	2	1	2019-03-30 11:34:00.000000	home
15	2	1	2019-03-30 11:41:00.000000	view
16	2	1	2019-03-30 11:45:00.000000	view
17	2	1	2019-03-30 11:52:00.000000	checkout
18	2	1	2019-03-30 11:58:00.000000	view

Lab 18: nPath FILTER Argument (cont.)

The (optional) FILTER argument lets you impose additional filters on the pattern matches being found by nPath. This clause lets you compare the current value of a row with a value from an unknown number of rows forward or behind (LAG/LEAD can't do this)

```
-- Without 'FILTER', output = 3 rows
SELECT * FROM nPath
(ON TRNG_TDU_TD01.clickstream
 PARTITION BY userid, sessionid
 ORDER BY clickts
 USING
 MODE(nonoverlapping)
 SYMBOLS(page = 'home' as home,
                page <> 'home' and page <> 'checkout' as view1,
                page = 'checkout' as checkout)
 PATTERN('home.view1*.checkout')
 RESULT(first(userid of any(home, checkout, view1)) as userid,
              first(sessionid of any (home, checkout, view1)) as sessionid,
              count(* of any (home, checkout, view1)) as cnt,
              first(clickts of any(home)) as first_home,
              last(clickts of any(checkout)) as last_checkout));
```

Output

	userid	sessionid	cnt	first_home	last_checkout
1	1	1	4	2019-02-27 10:15:00.000000	2019-02-27 10:18:00.000000
2	2	1	4	2019-03-10 11:34:00.000000	2019-03-10 11:52:00.000000
3	2	1	4	2019-03-30 11:34:00.000000	2019-03-30 11:52:00.000000

Lab 18: nPath FILTER Argument (cont.)


```
-- Query: Find users who visited the CHECKOUT page within 10 minutes of HOME page
-- With 'FILTER', Output = 1 row
SELECT * FROM nPath
(ON TRNG_TDU_TD01.clickstream
 PARTITION BY userid, sessionid
 ORDER BY clickts
 USING
 MODE (NONOVERLAPPING)
 SYMBOLS (page = 'home' AS home,
                page NE ' home' AND page NE 'checkout' AS view1,
                page = 'checkout' AS checkout)
 PATTERN ('home.view1*.checkout')
 RESULT (FIRST(userid of ANY(home, checkout, view1)) AS userid,
         FIRST (sessionid of ANY(home, checkout, view1)) AS sessionid,
         COUNT (* of any(home, checkout, view1)) AS cnt,
         FIRST (clickts of ANY(home)) AS firsthome,
         LAST (clickts of ANY(checkout)) AS lastcheckout)
 FILTER (FIRST (clickts + interval'10' minute OF ANY (home)) > FIRST (clickts of any(checkout)))
) AS dt;
```

Output

	userid	sessionid	cnt	firsthome	lastcheckout
1	1	1	4	2019-02-27 10:15:00.000000	2019-02-27 10:18:00.000000

Current Topic – Vantage Analyst

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
- Symbols
- Mode
- Pattern with Symbols
- Result
- Daisy Chaining, Multiple Input, & Filter Argument
- **Vantage Analyst**
- Hackathon & Review

Teradata Vantage Analyst Introduction

- Vantage Analyst is a Web-based application
- It is designed for the Business Analyst
- Vantage Analyst is a point-and-click interface which builds code for you behind the scenes
- To do **nPath Tree Charts** in Vantage Analyst, there are a number of rules you must follow:
  1. Underlying Table name must have '_EVENTS'
  2. At minimum, must have the following Column names in Table
     a. Entity_id
     b. Datestamp
     c. Event

Lab 19a: Vantage Analyst Tree Chart for Paths Leading to 'BME': View the Data

First, let's view the Table contents. This is a slightly different dataset than prior Lab. Data has to be 'SESSIONZEd' only

```
SELECT * FROM bank_events
ORDER BY entity_id, datestamp;
```

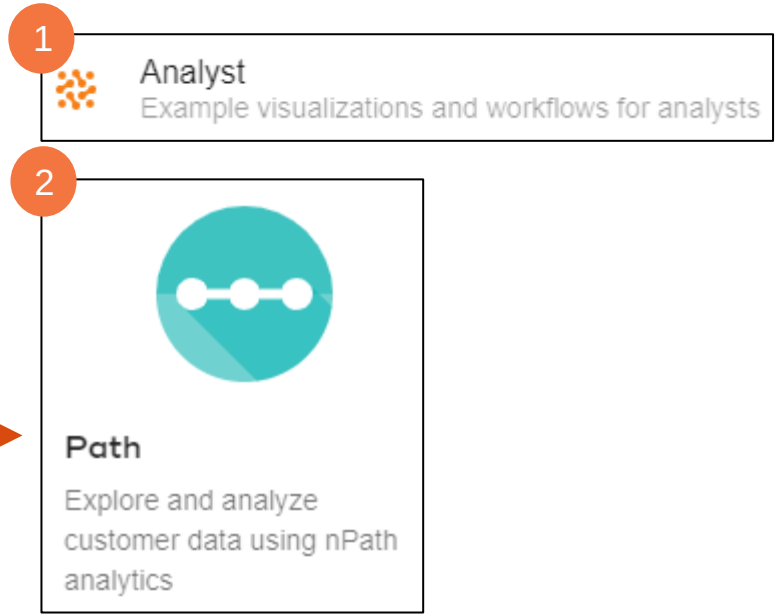
	entity_id	datestamp	event	sessionid
1	32	2004-04-16 14:18:14.000000	ACCOUNT SUMMARY	4
2	32	2004-04-16 14:20:34.000000	BILL MANAGER FORM	4
3	32	2004-04-16 14:22:52.000000	BILL MANAGER ENROLLMENT	4
4	75	2004-05-10 23:22:49.000000	VIEW DEPOSIT DETAILS	1
5	75	2004-05-10 23:25:37.000000	ONLINE STATEMENT ENROLLMENT	1
6	75	2004-05-10 23:29:09.000000	FAQ	1
7	75	2004-05-10 23:32:23.000000	BILL MANAGER FORM	1
8	75	2004-05-10 23:34:33.000000	BILL MANAGER ENROLLMENT	1
9	77	2004-04-10 22:16:37.000000	ACCOUNT SUMMARY	1
10	77	2004-04-10 22:17:42.000000	ONLINE STATEMENT ENROLLMENT	1

Lab 19b: Vantage Analyst Tree Chart for Paths Leading to 'BME'

Follow along with Instructor as we build Chart using existing Table similar to last Lab

To get back to 'Home' screen, click 'Teradata' logo near top left of the window

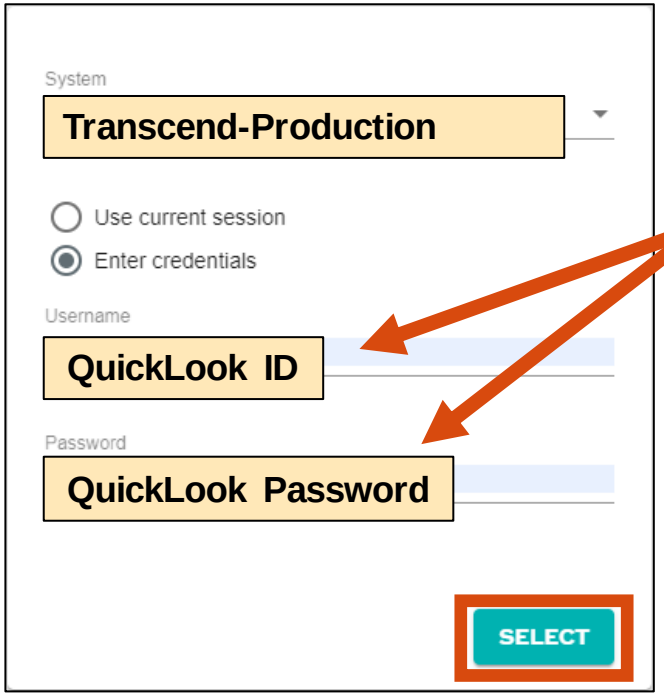
Next click 'Analyst', then 'path' Widget to get started



'BME' = Bill Management Enrollment

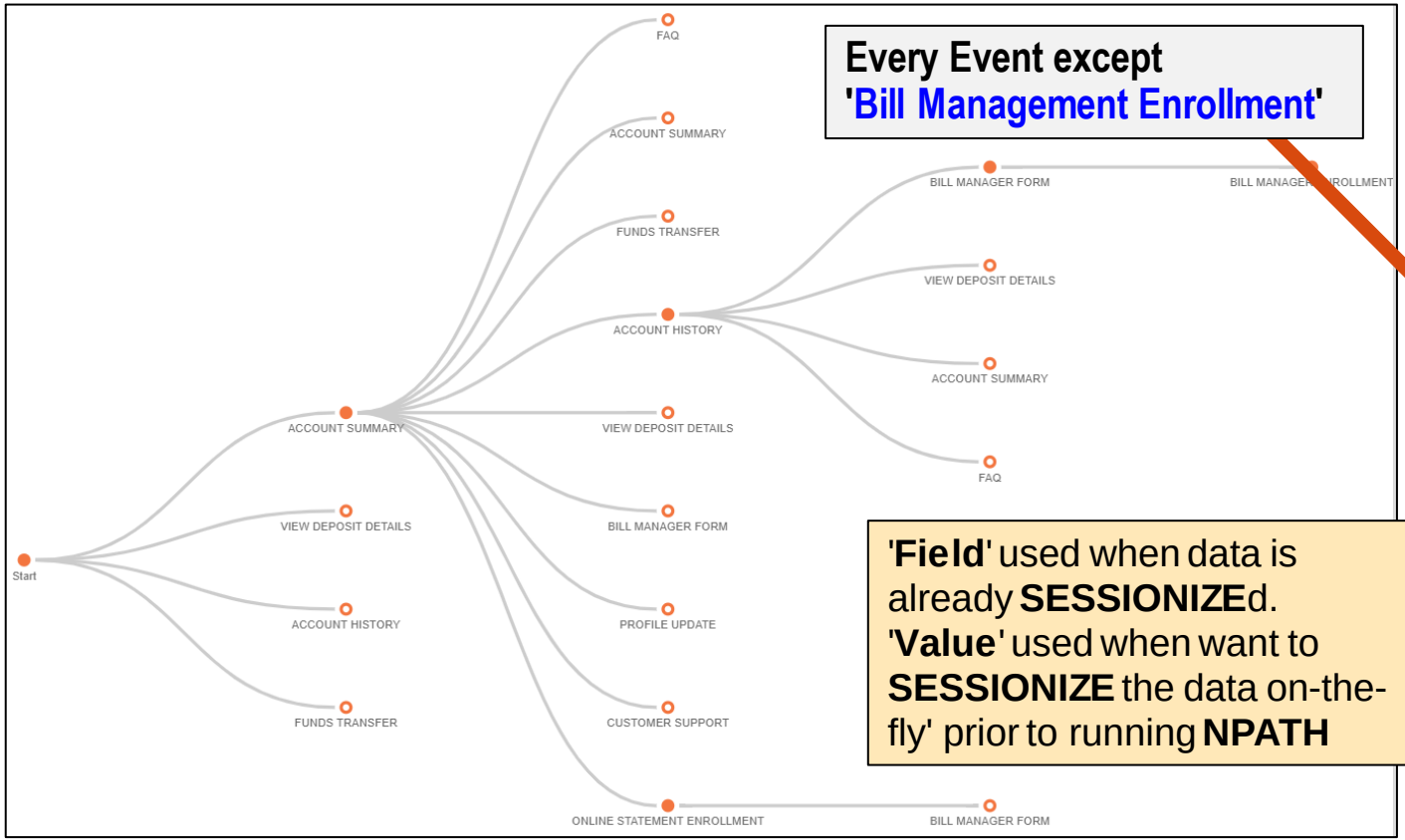
Lab 19b: Vantage Analyst Tree Chart for Paths Leading to 'BME' (cont.)

Select system



If possible, Use Current Session (default). If login fails, then use QuickLook ID

Lab 19b: Vantage Analyst Tree Chart for Paths Leading to 'BME' (cont.)



Every Event except 'Bill Management Enrollment'

'**Field**' used when data is already **SESSIONIZE**d.
'**Value**' used when want to **SESSIONIZE** the data on-the-fly' prior to running **NPATH**

Data source

Event database
TRNG_TDU_TD01

Event table
bank_events

Event A
ACCOUNT HISTORY, ACCOUN.
Starting anchor

Event B
BILL MANAGER ENROLLMENT
Ending anchor

Session

FIELD | VALUE

Session column
sessionid

SHOW SQL | RUN

Current Topic – Hackathon and Review

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
- Symbols
- Mode
- Pattern with Symbols
- Result
- Daisy Chaining, Multiple Input, & Filter Argument
- Vantage Analyst
- **Hackathon & Review**

Hackathon 1: Airlines Hackathon

Query: I live in ATL and want to travel to DEN. A direct flight costs too much. Buy cheaper ticket that passes thru DEN to any final dest (ie: ATL>DEN> ???)

Find all 2-leg flights where origin=ATL that pass thru DEN onto any final destination using same jet

```
SELECT uniquecarrier, flightnum, tailnum, origin, dest, yr, mnth, dayofmonth, crsdeptime
FROM airlines SAMPLE 1;
```

uniquecarrier	flightnum	tailnum	origin	dest	yr	mnth	dayofmonth	crsdeptime
UA	1492	N492UA	LAX	LAS	2008	1	27	2230

```
SELECT * FROM nPath (on
(SELECT yr, mnth, dayofmonth, uniquecarrier, flightnum, tailnum, origin, dest,
 CAST(Yr*10000+MNTH*100+DAYOFMONTH - 19000000 AS DATE) as mydt FROM airlines)
 PARTITION BY
 USING
 MODE
 SYMBOLS
 PATTERN
 RESULT
                                                                              );
```

Output

	carrier	flightnum	origin	pass_thru	finaldest
3	DL	781	ATL	DEN	BWI
4	F9	551	ATL	DEN	SAN
5	FL	309	ATL	DEN	BWI
6	UA	369	ATL	DEN	SFO

Hackathon 2: Visits before First >10% Discount & Visits after First >10% Discount?


```
SELECT * FROM nPath
(ON sales_fact
PARTITION BY customer_id
ORDER BY sales_date
USING
MODE(                    )
PATTERN(                                  )
SYMBOLS(discount_amount < .10 as NODISCOUNT,
        discount_amount >= .10 as DISCOUNT,
        '       A)
RESULT (      (customer_id of DISCOUNT) as customer_id
      ,       (sales_date of DISCOUNT) as discount_date
      ,       (discount_amount of DISCOUNT) as first_large_discount
      , COUNT(      NODISCOUNT) AS pre_discount_visits
      , COUNT(      A) as post_discount_visits));
```

Output

	customer_id	discount_date	first_large_discoun	pre_discount_visits	post_discount_visits
1	273	2008-01-01	0.143	0	63
2	189	2008-01-14	0.152	1	96
3	310	2008-01-05	0.111	1	154
4	192	2008-01-07	0.199	0	300
5	109	2008-01-02	0.118	1	379
6	6	2008-02-11	0.171	0	4

Row in table is defined as 'Visit'

Re-write last 2 lines of code to show summed 'sales_quantity' for both pre/post discount

Hackathon 2: Sales_quantity before First >10% Discount & Sales_quantity after First >10% Discount?


```
SELECT * FROM nPath
(ON sales_fact
PARTITION BY customer_id
ORDER BY sales_date
USING
MODE(NONOVERLAPPING)
PATTERN('^(NODISCOUNT*).(DISCOUNT).(A*)')
SYMBOLS(discount_amount < .10 as NODISCOUNT,
        discount_amount >= .10 as DISCOUNT,
        TRUE as A)
RESULT (FIRST(customer_id of DISCOUNT) as customer_id
      , FIRST(sales_date of DISCOUNT) as discount_date
      , FIRST(discount_amount of DISCOUNT) as first_large_discount
      ,
      ,
```

Output

customer_id	discount_date	first_large_discount	nodisc_sum_qty	disc_sum_qty
1	2008-01-23	0.141	null	175
2	2008-01-02	0.112	null	150
3	2008-01-10	0.17	null	736
4	2008-07-13	0.197	12	21
5	2008-01-03	0.183	null	1310
6	2008-02-11	0.171	null	36
7	2008-01-02	0.196	null	1135

Row in table is defined as 'Visit'

Re-write last 2 lines of code to show summed 'sales_quantity' for both pre/post discount

Hackathon 3: Baseball Losing Streaks

The following exercise is intended to provide you with further practice on using the **nPath** function. There is no 'right' or 'wrong' answer, as there may be multiple viable ways to arrive at a meaningful answer-set. The intent is for you to become comfortable writing queries that use **nPath**

1. Run an **nPath** query on the **teams_prior2012** table, which shows baseball team statistics FROM 1871 to 2011. Your goal is to display how many years elapsed before a team won their division title, focusing on teams that had losing streaks greater than or equal to 10 years before winning a division title. **Hint:** The **divwin** column displays whether the team won the division or not. Each team has a unique identifier, displayed in the **teamid** column. Long-names appear in the **name** column
2. Things to think about follow:
   - What is the *nature* of the underlying data? Data types? Number of rows? What is it showing?
   - How should the data be partitioned and ordered?
   - What should the defined **PATTERN** be?
   - Etc.

Hackathon 3: Baseball Losing Streaks (Possible Answer)


```
SELECT * FROM teams_prior2012
sample randomized allocation 200;

-- find losing streaks >= 10 years before a DIVWIN.
SELECT team, first_div_losing_year,
       first_div_winning_year, years_before_div_win
FROM nPath
(ON teams_prior2012
PARTITION BY teamid ORDER BY yearid
USING
Mode (NONOVERLAPPING)
Symbols (divwin = 'N' as N, divwin= 'Y' as Y)
Pattern ('N+.Y')
Result (Last (name of N) as team, First (yearid of N)
         as first_div_losing_year,
        First (yearid of Y) as first_div_winning_year,
        Count(distinct yearid of any(N)) as
         years_before_div_win)) AS dt
GROUP BY team, first_div_losing_year,
   first_div_winning_year, years_before_div_win
HAVING years_before_div_win >= 10
ORDER BY years_before_div_win DESC, team ASC;
```

Input Data (subset)

	yearid	leagueid	teamid	franchid	divid	ranked
1	1967	NL	CIN	CIN	null	4
2	1924	AL	PHA	OAK	null	5
3	1956	NL	BRO	LAD	null	1
4	1883	AA	SL4	STL	null	2
5	1949	AL	SLA	BAL	null	7

nPath Output (subset)

	team	first_div_losing_year	first_div_winning_year	years_before_div_win
1	Detroit Tigers	1995	2011	16
2	Minnesota Twins	1971	1987	16
3	Chicago Cubs	1969	1984	15
4	San Diego Padres	1969	1984	15
5	San Francisco Giants	1972	1987	15
6	Chicago White Sox	1969	1983	14
7	Cincinnati Reds	1996	2010	14
8	Milwaukee Brewers	1998	2011	13
9	St. Louis Cardinals	1969	1982	13
10	Atlanta Braves	1970	1982	12

Summary

In this module, you learned how to:

- Describe what the **nPath** function does
- Describe typical use cases for **nPath**
- Write **nPath** queries
- Interpret the output of **nPath** queries

Thank you.

teradata.