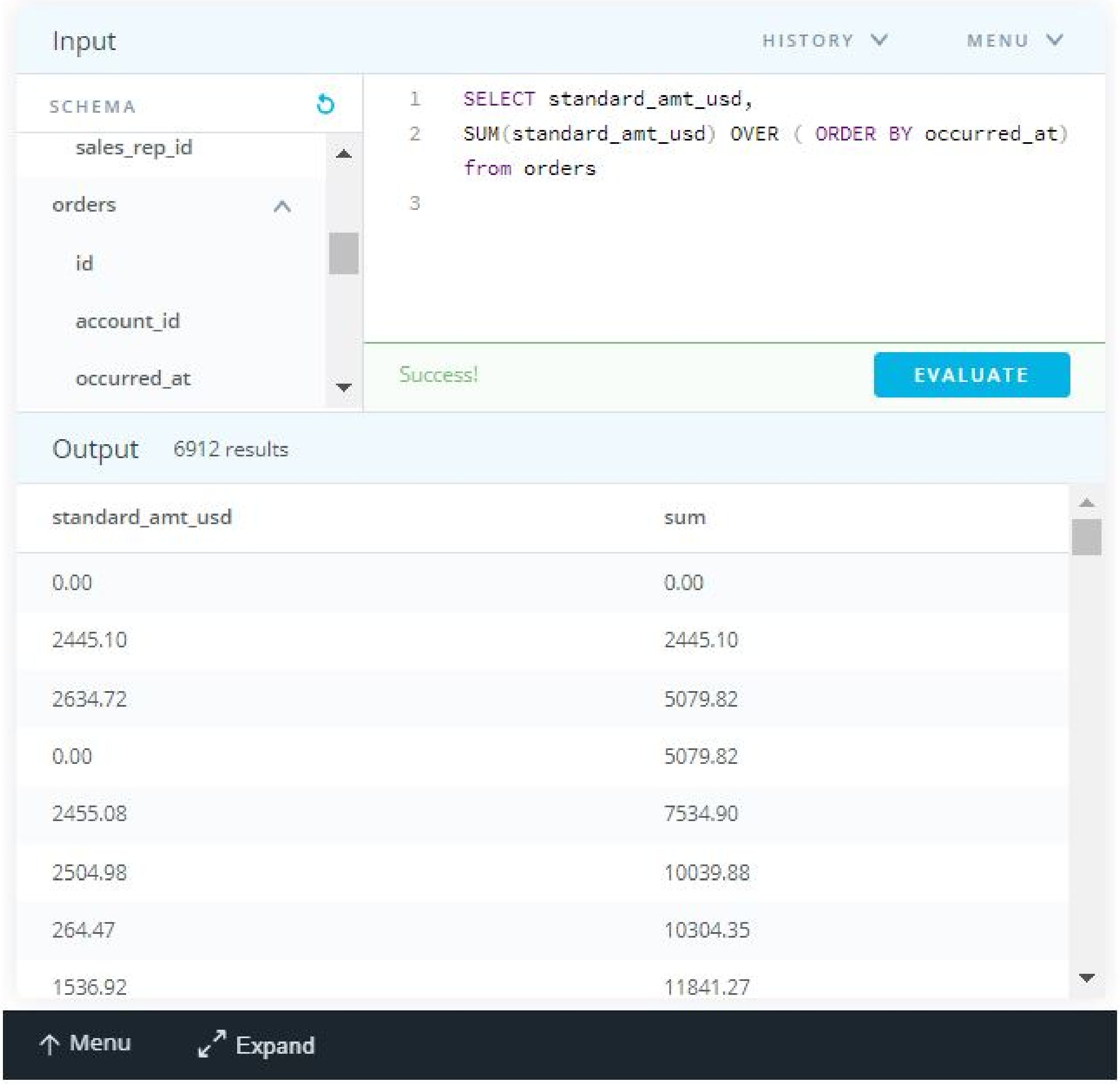
Creating a Running Total Using Window Functions

Using Derek's previous video as an example, create another running total. This time, create a running total of `standard_amt_usd` (in the `orders` table) over order time with no date truncation. Your final table should have two columns: one with the amount being added for each new row, and a second with the running total.

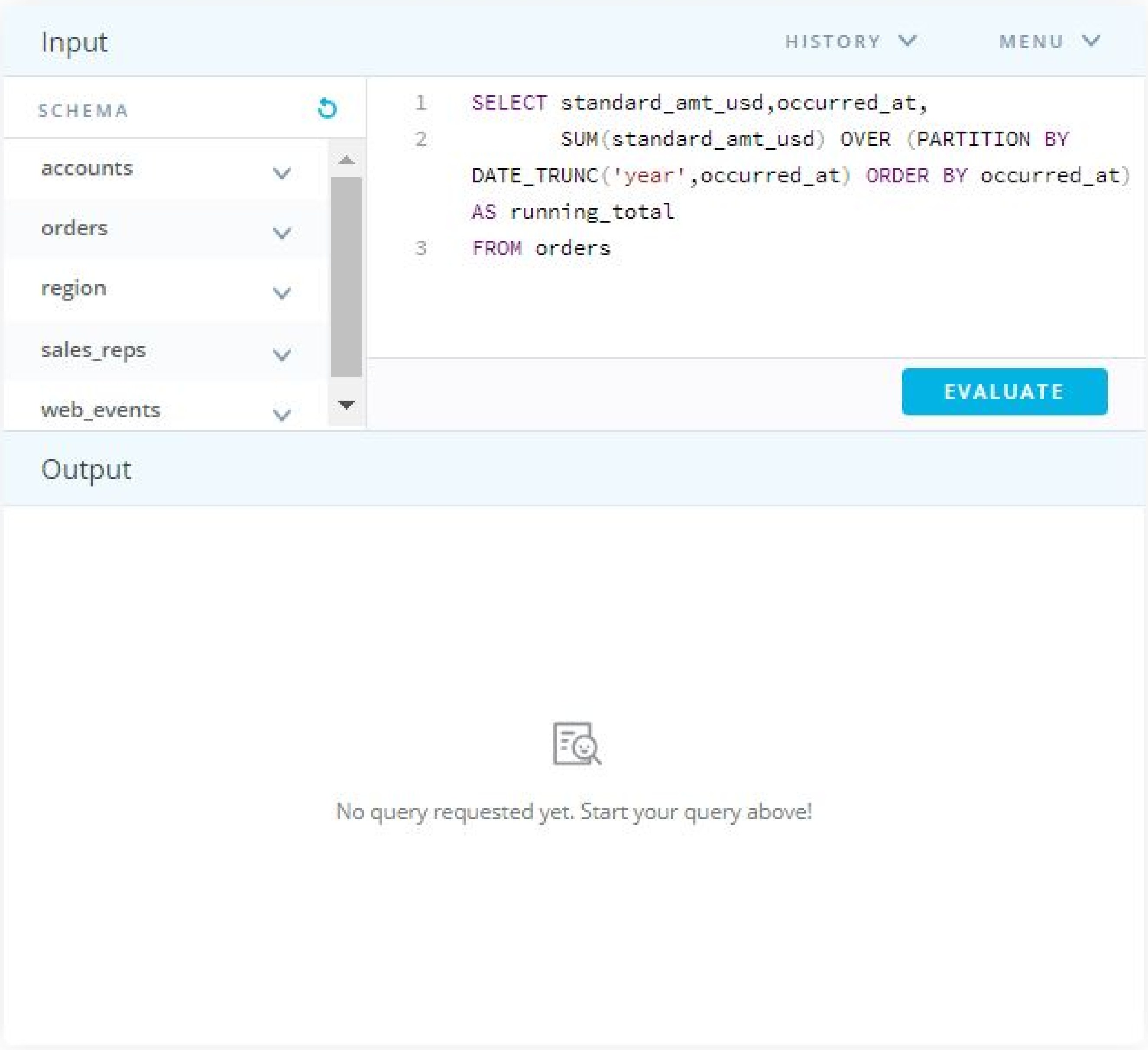


```
SELECT standard_amt_usd,
SUM(standard_amt_usd) OVER ( ORDER BY occurred_at)
from orders
```

standard_amt_usd	sum
0.00	0.00
2445.10	2445.10
2634.72	5079.82
0.00	5079.82
2455.08	7534.90
2504.98	10039.88
264.47	10304.35
1536.92	11841.27

Creating a *Partitioned* Running Total Using Window Functions

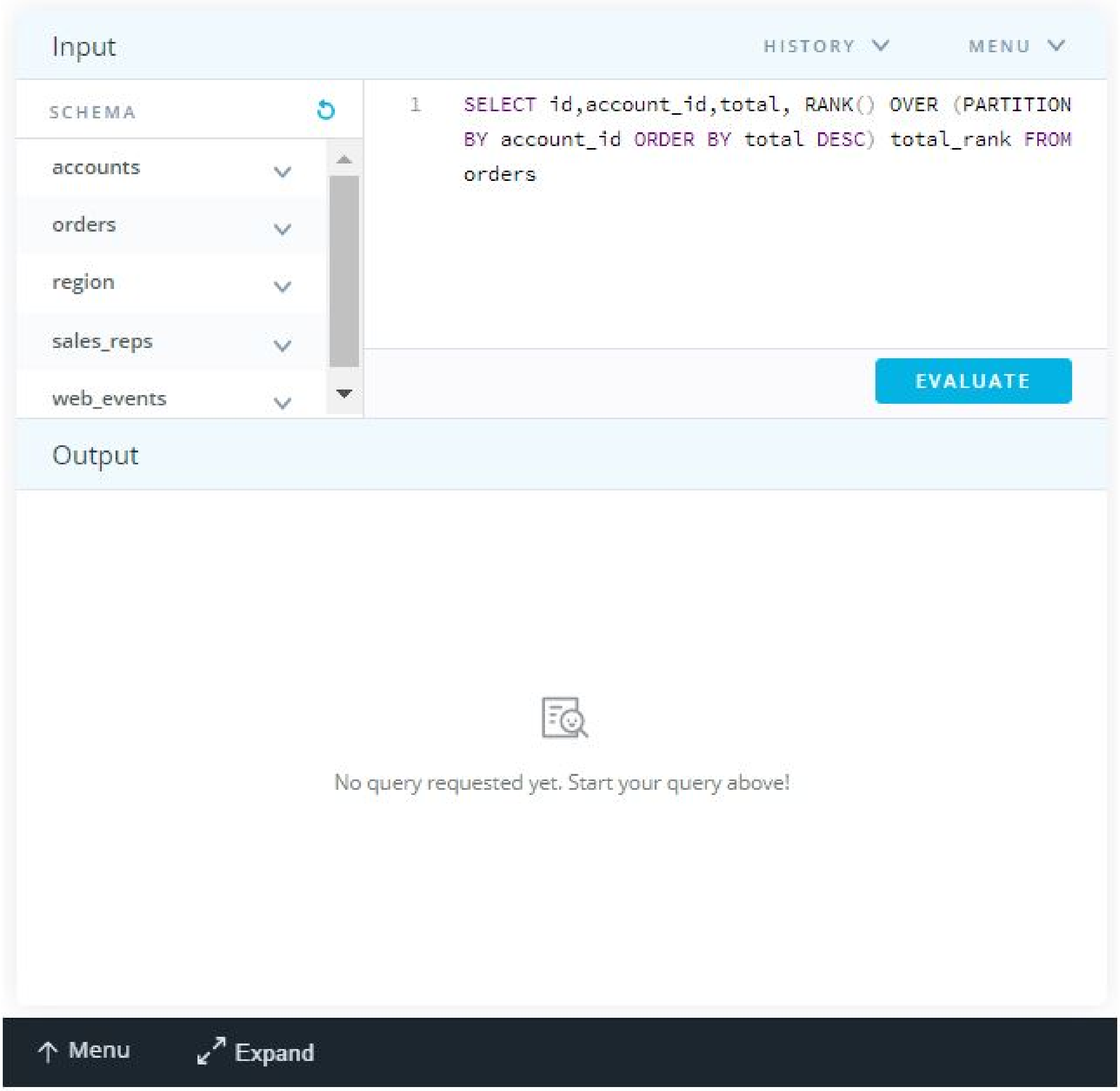
Now, modify your query from the previous quiz to include partitions. Still create a running total of `standard_amt_usd` (in the `orders` table) over order time, but this time, date truncate `occurred_at` by year and partition by that same year-truncated `occurred_at` variable. Your final table should have three columns: One with the amount being added for each row, one for the truncated date, and a final column with the running total within each year.



```sql
SELECT standard_amt_usd,occurred_at,
       SUM(standard_amt_usd) OVER (PARTITION BY DATE_TRUNC('year',occurred_at) ORDER BY occurred_at) AS running_total
FROM orders
```

Ranking Total Paper Ordered by Account

Select the `id`, `account_id`, and `total` variable from the `orders` table, then create a column called `total_rank` that ranks this total amount of paper ordered (from highest to lowest) *for each account* using a partition. Your final table should have these four columns.



```
SELECT id,account_id,total, RANK() OVER (PARTITION
BY account_id ORDER BY total DESC) total_rank FROM
orders
```

No query requested yet. Start your query above!

WINDOW FUNCTIONS

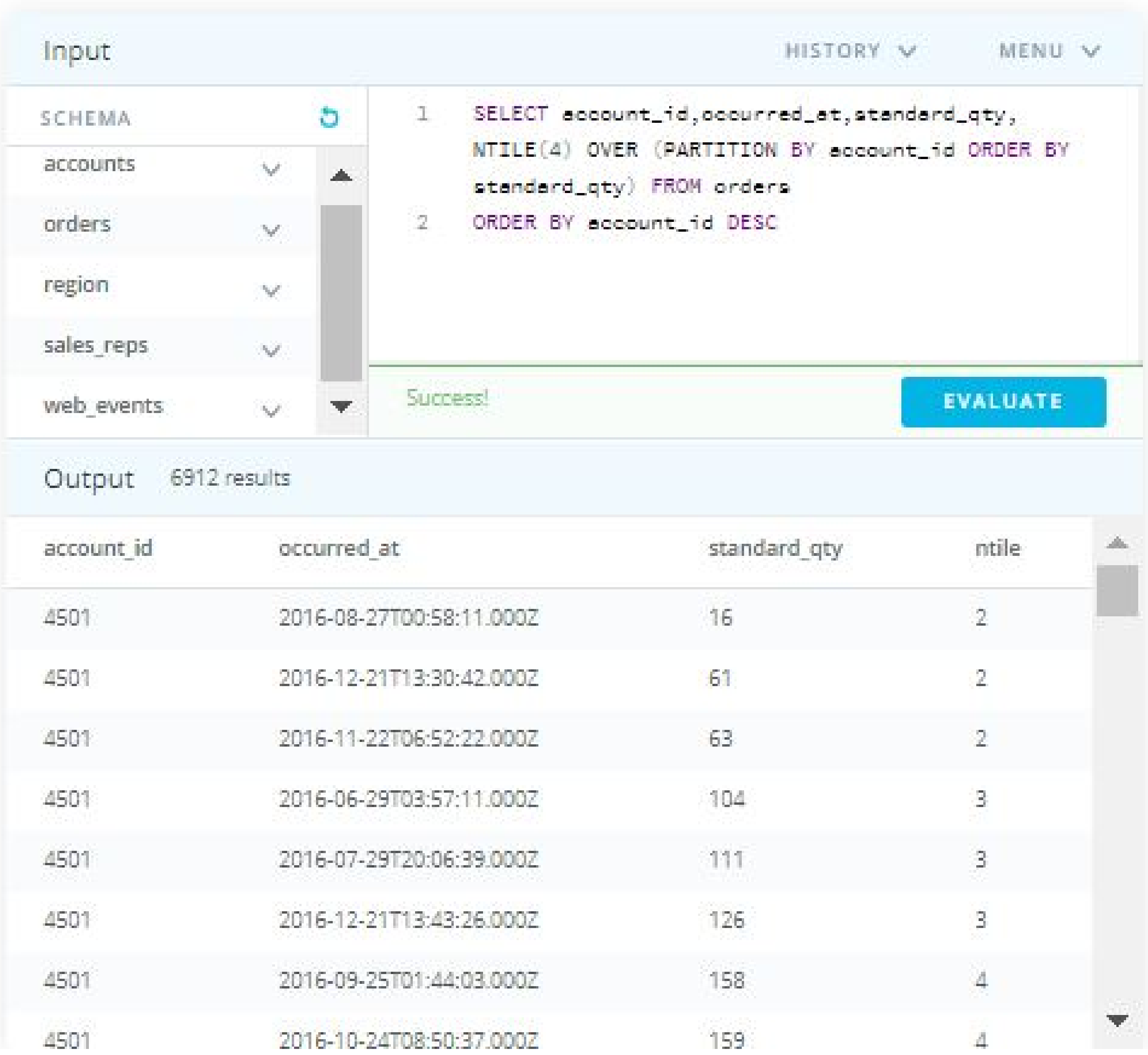
```
SELECT id,

       account_id,

       DATE_TRUNC('year',occurred_at) AS year,

       DENSE_RANK() OVER account_year_window AS dense_rank,

       total_amt_usd,

       SUM(total_amt_usd) OVER account_year_window AS sum_total_amt_usd,

       COUNT(total_amt_usd) OVER account_year_window AS count_total_amt_usd,

       AVG(total_amt_usd) OVER account_year_window AS avg_total_amt_usd,

       MIN(total_amt_usd) OVER account_year_window AS min_total_amt_usd,

       MAX(total_amt_usd) OVER  account_year_window AS max_total_amt_usd

FROM orders

WINDOW account_year_window AS (PARTITION BY account_id ORDER BY
DATE_TRUNC('year',occurred_at))
```

1. Use the `NTILE` functionality to divide the accounts into 4 levels in terms of the amount of `standard_qty` for their orders. Your resulting table should have the `account_id`, the `occurred_at` time for each order, the total amount of `standard_qty` paper purchased, and one of four levels in a `standard_quartile` column.

2. Use the `NTILE` functionality to divide the accounts into two levels in terms of the amount of `gloss_qty` for their orders. Your resulting table should have the `account_id`, the `occurred_at` time for each order, the total amount of `gloss_qty` paper purchased, and one of two levels in a `gloss_half` column.

3. Use the `NTILE` functionality to divide the orders for each account into 100 levels in terms of the amount of `total_amt_usd` for their orders. Your resulting table should have the `account_id`, the `occurred_at` time for each order, the total amount of `total_amt_usd` paper purchased, and one of 100 levels in a `total_percentile` column.

Note: To make it easier to interpret the results, order by the account_id in each of the queries.



```
SELECT account_id,occurred_at,standard_qty,
NTILE(4) OVER (PARTITION BY account_id ORDER BY
standard_qty) FROM orders
ORDER BY account_id DESC
```

Success!

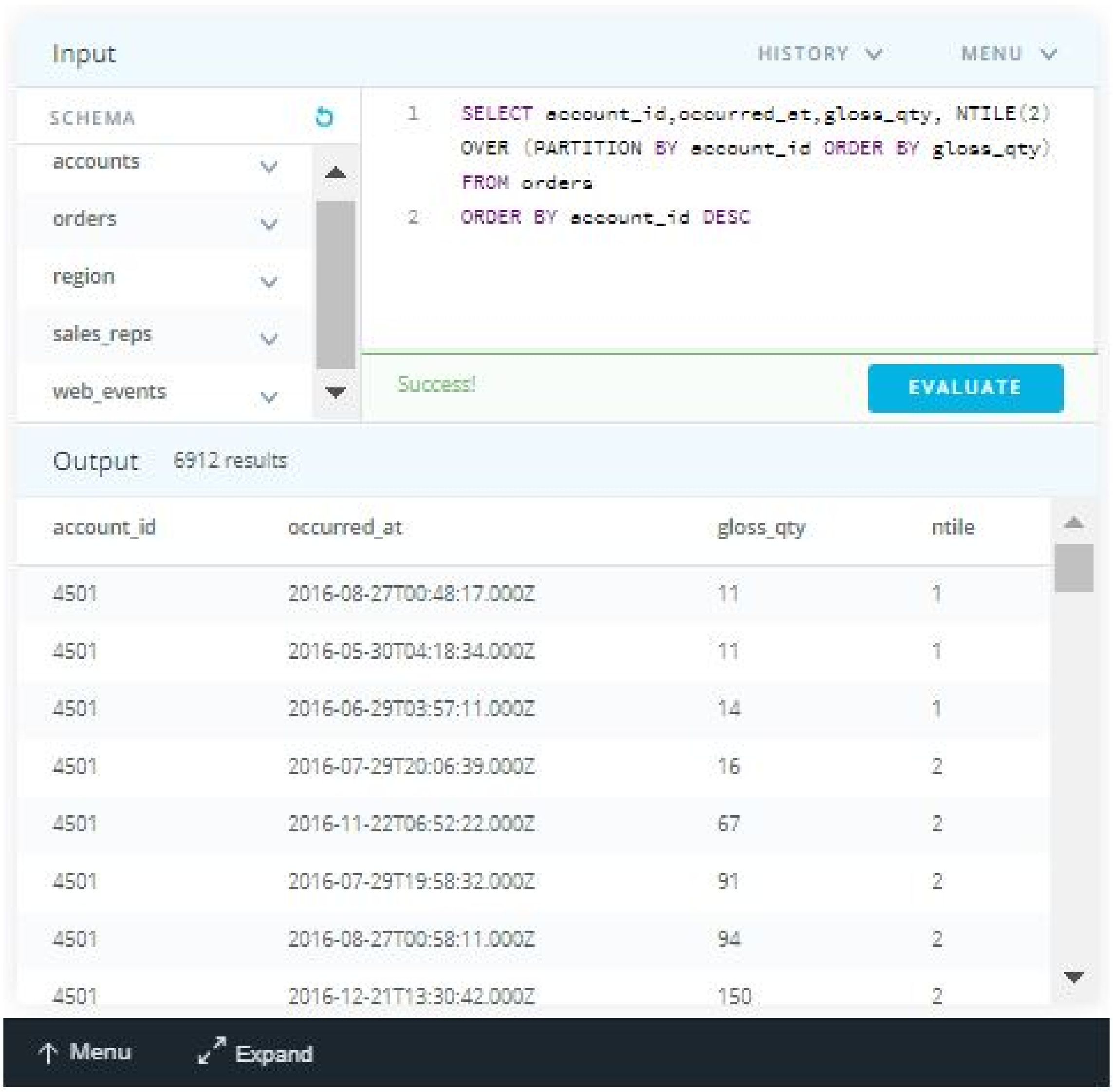
6912 results

account_id	occurred_at	standard_qty	ntile
4501	2016-08-27T00:58:11.000Z	16	2
4501	2016-12-21T13:30:42.000Z	61	2
4501	2016-11-22T06:52:22.000Z	63	2
4501	2016-06-29T03:57:11.000Z	104	3
4501	2016-07-29T20:06:39.000Z	111	3
4501	2016-12-21T13:43:26.000Z	126	3
4501	2016-09-25T01:44:03.000Z	158	4
4501	2016-10-24T08:50:37.000Z	159	4

2. Use the `NTILE` functionality to divide the accounts into two levels in terms of the amount of `gloss_qty` for their orders. Your resulting table should have the `account_id`, the `occurred_at` time for each order, the total amount of `gloss_qty` paper purchased, and one of two levels in a `gloss_half` column.

3. Use the `NTILE` functionality to divide the orders for each account into 100 levels in terms of the amount of `total_amt_usd` for their orders. Your resulting table should have the `account_id`, the `occurred_at` time for each order, the total amount of `total_amt_usd` paper purchased, and one of 100 levels in a `total_percentile` column.

Note: To make it easier to interpret the results, order by the account_id in each of the queries.



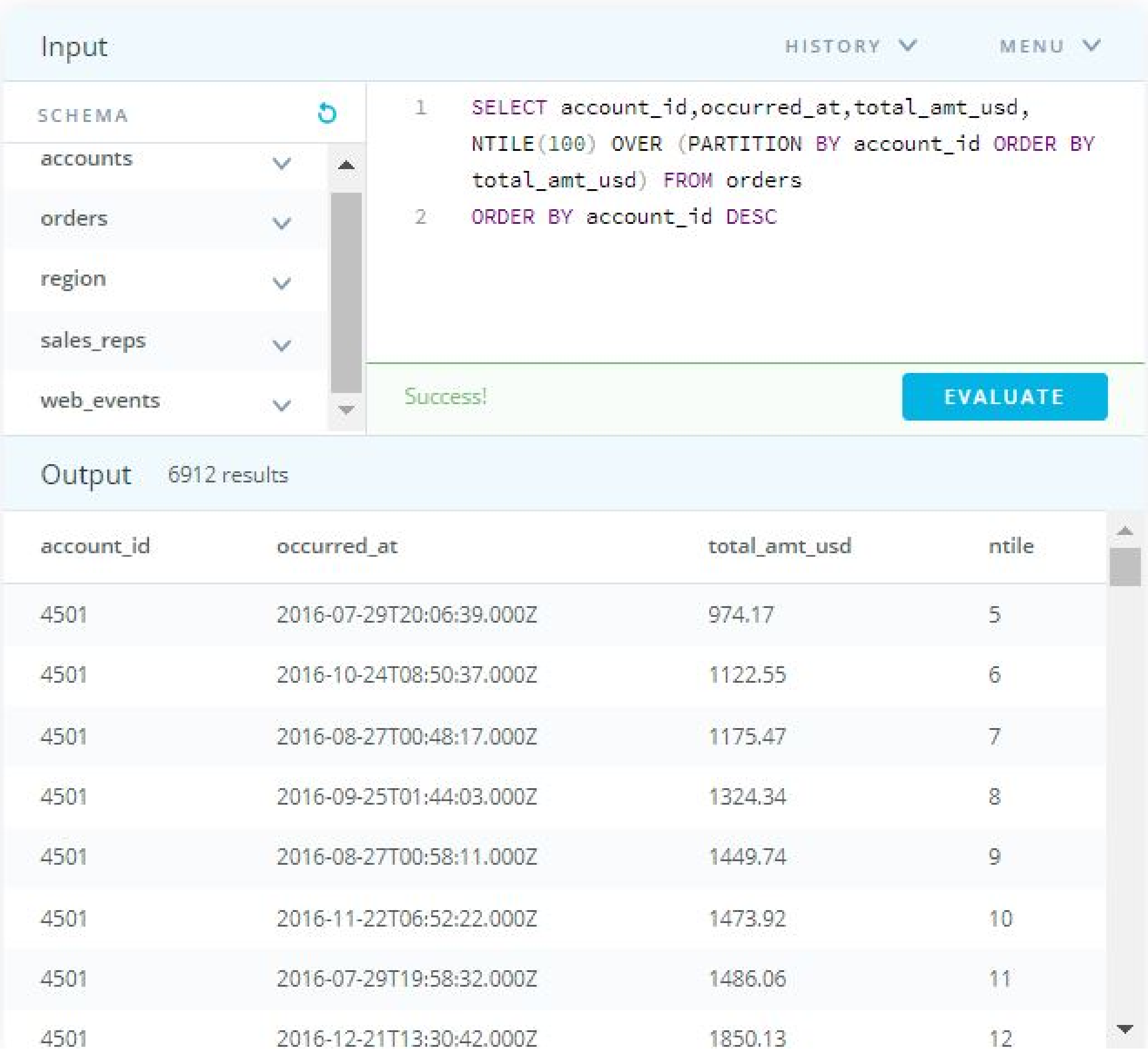
```
SELECT account_id,occurred_at,gloss_qty, NTILE(2)
OVER (PARTITION BY account_id ORDER BY gloss_qty)
FROM orders
ORDER BY account_id DESC
```

Output 6912 results

account_id	occurred_at	gloss_qty	ntile
4501	2016-08-27T00:48:17.000Z	11	1
4501	2016-05-30T04:18:34.000Z	11	1
4501	2016-06-29T03:57:11.000Z	14	1
4501	2016-07-29T20:06:39.000Z	16	2
4501	2016-11-22T06:52:22.000Z	67	2
4501	2016-07-29T19:58:32.000Z	91	2
4501	2016-08-27T00:58:11.000Z	94	2
4501	2016-12-21T13:30:42.000Z	150	2

3. Use the `NTILE` functionality to divide the orders for each account into 100 levels in terms of the amount of `total_amt_usd` for their orders. Your resulting table should have the `account_id`, the `occurred_at` time for each order, the total amount of `total_amt_usd` paper purchased, and one of 100 levels in a `total_percentile` column.

Note: To make it easier to interpret the results, order by the account_id in each of the queries.



```
SELECT account_id,occurred_at,total_amt_usd,
NTILE(100) OVER (PARTITION BY account_id ORDER BY
total_amt_usd) FROM orders
ORDER BY account_id DESC
```

account_id	occurred_at	total_amt_usd	ntile
4501	2016-07-29T20:06:39.000Z	974.17	5
4501	2016-10-24T08:50:37.000Z	1122.55	6
4501	2016-08-27T00:48:17.000Z	1175.47	7
4501	2016-09-25T01:44:03.000Z	1324.34	8
4501	2016-08-27T00:58:11.000Z	1449.74	9
4501	2016-11-22T06:52:22.000Z	1473.92	10
4501	2016-07-29T19:58:32.000Z	1486.06	11
4501	2016-12-21T13:30:42.000Z	1850.13	12