Pattern Matching (MATCH_RECOGNIZE)

Introduced in Oracle 8i, Analytic Functions, also known as windowing functions, allow developers to perform tasks in SQL that were previously confined to procedural languages. Oracle 12c has added the MATCH_RECOGNIZE clause into the analytic function syntax to make pattern matching from SQL simpler. This article gives a flavour of what can be done using the MATCH_RECOGNIZE clause, but you will need to refer to the documentation to understand the true level of complexity possible.

Setup

```
CREATE TABLE sales_history (  
id NUMBER,  
product VARCHAR2(20),  
tstamp TIMESTAMP,  
units_sold NUMBER,  
CONSTRAINT sales_history_pk PRIMARY KEY (id)  
);

ALTER SESSION SET nls_timestamp_format = 'DD-MON-YYYY';

INSERT INTO sales_history VALUES ( 1, 'TWINKIES', '01-OCT-2014', 17);  
INSERT INTO sales_history VALUES ( 2, 'TWINKIES', '02-OCT-2014', 19);  
INSERT INTO sales_history VALUES ( 3, 'TWINKIES', '03-OCT-2014', 23);  
INSERT INTO sales_history VALUES ( 4, 'TWINKIES', '04-OCT-2014', 23);  
INSERT INTO sales_history VALUES ( 5, 'TWINKIES', '05-OCT-2014', 16);  
INSERT INTO sales_history VALUES ( 6, 'TWINKIES', '06-OCT-2014', 10);  
INSERT INTO sales_history VALUES ( 7, 'TWINKIES', '07-OCT-2014', 14);  
INSERT INTO sales_history VALUES ( 8, 'TWINKIES', '08-OCT-2014', 16);  
INSERT INTO sales_history VALUES ( 9, 'TWINKIES', '09-OCT-2014', 15);  
INSERT INTO sales_history VALUES (10, 'TWINKIES', '10-OCT-2014', 17);  
INSERT INTO sales_history VALUES (11, 'TWINKIES', '11-OCT-2014', 23);  
INSERT INTO sales_history VALUES (12, 'TWINKIES', '12-OCT-2014', 30);  
INSERT INTO sales_history VALUES (13, 'TWINKIES', '13-OCT-2014', 31);  
INSERT INTO sales_history VALUES (14, 'TWINKIES', '14-OCT-2014', 29);  
INSERT INTO sales_history VALUES (15, 'TWINKIES', '15-OCT-2014', 25);  
INSERT INTO sales_history VALUES (16, 'TWINKIES', '16-OCT-2014', 21);  
INSERT INTO sales_history VALUES (17, 'TWINKIES', '17-OCT-2014', 35);
```
The following query shows the pattern of the data, which we will refer to later.

```
ALTER SESSION SET nls_timestamp_format = 'DD-MON-YYYY';

SELECT id, product, tstamp, units_sold, RPAD('=', units_sold, '=') AS graph
FROM sales_history
ORDER BY id;
```

<table>
<thead>
<tr>
<th>ID</th>
<th>PRODUCT</th>
<th>TSTAMP</th>
<th>UNITS_SOLD</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TWINKIES</td>
<td>01-OCT-2014</td>
<td>17</td>
<td>########################</td>
</tr>
<tr>
<td>2</td>
<td>TWINKIES</td>
<td>02-OCT-2014</td>
<td>19</td>
<td>########################</td>
</tr>
<tr>
<td>3</td>
<td>TWINKIES</td>
<td>03-OCT-2014</td>
<td>23</td>
<td>########################</td>
</tr>
<tr>
<td>4</td>
<td>TWINKIES</td>
<td>04-OCT-2014</td>
<td>23</td>
<td>########################</td>
</tr>
<tr>
<td>5</td>
<td>TWINKIES</td>
<td>05-OCT-2014</td>
<td>16</td>
<td>########################</td>
</tr>
<tr>
<td>6</td>
<td>TWINKIES</td>
<td>06-OCT-2014</td>
<td>10</td>
<td>#######</td>
</tr>
<tr>
<td>7</td>
<td>TWINKIES</td>
<td>07-OCT-2014</td>
<td>14</td>
<td>#######</td>
</tr>
<tr>
<td>8</td>
<td>TWINKIES</td>
<td>08-OCT-2014</td>
<td>16</td>
<td>#######</td>
</tr>
<tr>
<td>9</td>
<td>TWINKIES</td>
<td>09-OCT-2014</td>
<td>15</td>
<td>#######</td>
</tr>
<tr>
<td>10</td>
<td>TWINKIES</td>
<td>10-OCT-2014</td>
<td>17</td>
<td>#######</td>
</tr>
<tr>
<td>11</td>
<td>TWINKIES</td>
<td>11-OCT-2014</td>
<td>23</td>
<td>########################</td>
</tr>
<tr>
<td>12</td>
<td>TWINKIES</td>
<td>12-OCT-2014</td>
<td>30</td>
<td>########################</td>
</tr>
<tr>
<td>13</td>
<td>TWINKIES</td>
<td>13-OCT-2014</td>
<td>31</td>
<td>########################</td>
</tr>
<tr>
<td>14</td>
<td>TWINKIES</td>
<td>14-OCT-2014</td>
<td>29</td>
<td>########################</td>
</tr>
<tr>
<td>15</td>
<td>TWINKIES</td>
<td>15-OCT-2014</td>
<td>25</td>
<td>########################</td>
</tr>
<tr>
<td>16</td>
<td>TWINKIES</td>
<td>16-OCT-2014</td>
<td>21</td>
<td>########################</td>
</tr>
<tr>
<td>17</td>
<td>TWINKIES</td>
<td>17-OCT-2014</td>
<td>35</td>
<td>########################</td>
</tr>
<tr>
<td>18</td>
<td>TWINKIES</td>
<td>18-OCT-2014</td>
<td>46</td>
<td>########################</td>
</tr>
</tbody>
</table>
If you check the data, you will see that there are 4 peaks

- 03-OCT-2014 (and 04-OCT-2014)
- 08-OCT-2014
- 13-OCT-2014
- 18-OCT-2014

Task: Write some code that will identify the beginning of a peak, the peak and the end of the decrease:

```
START_TSTAM PEAK_TSTAMP END_TSTAMP
----------- -----------  -----------
01-OCT-2014 03-OCT-2014 06-OCT-2014
06-OCT-2014 08-OCT-2014 09-OCT-2014
09-OCT-2014 13-OCT-2014 16-OCT-2014
16-OCT-2014 18-OCT-2014 20-OCT-2014
```

The output tells us that there were 4 distinct peaks/spikes in the sales, giving us the location of the start, peak and end of the pattern. Can the output be achieved using a single SQL statement only?

This can be achieved with an SQL statement using match_recognize. There is no need to write a complex program! Currently, only Oracle supports this clause which is art of the SQL standard. Other database vendors will most likely offer similar functionality in future.

```
ALTER SESSION SET nls_timestamp_format = 'DD-MON-YYYY';

SELECT *
FROM   sales_history MATCH_RECOGNIZE (PARTITION BY product ORDER BY tstamp MEASURES STRT.tstamp AS start_tstamp,
                                             LAST(UP.tstamp) AS peak_tstamp,
                                             LAST(DOWN.tstamp) AS end_tstamp,
                                             MATCH_NUMBER() AS mno
```
ONE ROW PER MATCH
AFTER MATCH SKIP TO LAST DOWN
PATTERN (STRT UP+ FLAT* DOWN+)
DEFINE
    UP AS UP.units_sold > PREV(UP.units_sold),
    FLAT AS FLAT.units_sold = PREV(FLAT.units_sold),
    DOWN AS DOWN.units_sold < PREV(DOWN.units_sold)
) MR
ORDER BY MR.product, MR.start_tstamp;

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>START_TSTAMP</th>
<th>PEAK_TSTAMP</th>
<th>END_TSTAMP</th>
<th>MNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>TWINKIES</td>
<td>01-OCT-2014</td>
<td>03-OCT-2014</td>
<td>06-OCT-2014</td>
<td>1</td>
</tr>
<tr>
<td>TWINKIES</td>
<td>06-OCT-2014</td>
<td>08-OCT-2014</td>
<td>09-OCT-2014</td>
<td>2</td>
</tr>
<tr>
<td>TWINKIES</td>
<td>09-OCT-2014</td>
<td>13-OCT-2014</td>
<td>16-OCT-2014</td>
<td>3</td>
</tr>
<tr>
<td>TWINKIES</td>
<td>16-OCT-2014</td>
<td>18-OCT-2014</td>
<td>20-OCT-2014</td>
<td>4</td>
</tr>
</tbody>
</table>

4 rows selected.

**Syntax Made Simple**

The pattern matching syntax includes a lot of options, which make it quite daunting at first. This section describes a very simplistic view of the syntax, allowing you to take your first steps. For a detailed description of the syntax, see the [documentation](#).

Data must be processed correctly and in a deterministic fashion. The PARTITION BY and ORDER BY clauses of all analytic functions are used to break the data up into groups and make sure it is ordered correctly within each group, so order-sensitive analytic functions work as expected. This is explained here. If no partitions are defined, it is assumed the whole result set is one big partition.

PARTITION BY product
ORDER BY tstamp

The MEASURES clause defines the column output that will be produced for each match.
Along with the MEASURES, you need to decide if you want to present all the rows that represent the match, or just summary information.

[ONE ROW | ALL ROWS] PER MATCH

The pattern that represents a match is defined using pattern variables, so it makes sense to look at those first. Pattern variables can use any non-reserved word associated with an expression. Two examples are given below.

```
DEFINE
   UP AS UP.units_sold > PREV(UP.units_sold),
   FLAT AS FLAT.units_sold = PREV(FLAT.units_sold),
   DOWN AS DOWN.units_sold < PREV(DOWN.units_sold)

DEFINE
   TWINKIES AS TWINKIES.product='TWINKIES',
   DINGDONGS AS DINGDONG.product='DINGDONGS',
   HOHOS AS HOHOS.product='HOHOS'
```

The pattern is then defined using regular expressions incorporating the pattern variables. Some examples are given below, but a full list of the possibilities is available from the documentation.

```
-- 1-Many increases, followed by 1-Many decreases in a value. A "V" shaped spike.
PATTERN (STRT UP+ DOWN+)

-- 1-Many increases, followed by a single decrease, then 1-Many increases. A single dip, during the rise.
PATTERN (STRT UP+ DOWN{1} UP+)

-- 1-5 Twinkies, followed by 1 DingDong, followed by 2 HoHos.
PATTERN(STRT TWINKIES{1,5} DINGDONGS(1) HOHOS{2})
```

The AFTER MATCH SKIP clause defines where the search is restarted from. Available options include the following.

- **AFTER MATCH SKIP TO NEXT ROW**: Search continues at the row following the start of the matched pattern.
- **AFTER MATCH SKIP PAST LAST ROW**: (Default) Search continues at the row following the end of the matched pattern.
- **AFTER MATCH SKIP TO FIRST pattern_variable**: Search continues from the first row relating to the pattern defined by the specified pattern variable.
• **AFTER MATCH SKIP TO LAST pattern_variable** : Search continues from the last row relating to the pattern defined by the specified pattern variable.
• **AFTER MATCH SKIP TO pattern_variable** : Equivalent of "AFTER MATCH SKIP TO LAST pattern_variable".

There are a number of functions that provide additional information about the displayed output.

• **MATCH_NUMBER()** : Sequential numbering of matches 1-N, indicating which output rows relate to which match.
• **CLASSIFIER()** : The pattern variable that applies to the output row. This only makes sense when all rows are displayed.

Navigation around the rows in a patterns is possible using the PREV, NEXT, FIRST and LAST functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREV(UP.units_sold)</td>
<td>-- Value of units_sold from previous row.</td>
</tr>
<tr>
<td>PREV(UP.units_sold, 2)</td>
<td>-- Value of units_sold from the row before the previous row (offset of 2 rows).</td>
</tr>
<tr>
<td>NEXT(UP.units_sold)</td>
<td>-- Value of units_sold from the next row.</td>
</tr>
<tr>
<td>NEXT(UP.units_sold, 2)</td>
<td>-- Value of units_sold from the row after the following row (offset of 2 rows).</td>
</tr>
<tr>
<td>FIRST(UP.units_sold)</td>
<td>-- First row in the pattern.</td>
</tr>
<tr>
<td>FIRST(UP.units_sold, 1)</td>
<td>-- Row following the first row (offset of 1 row).</td>
</tr>
<tr>
<td>LAST(UP.units_sold)</td>
<td>-- Last row in the pattern.</td>
</tr>
<tr>
<td>LAST(UP.units_sold, 1)</td>
<td>-- Row preceding the last row (offset of 1 row).</td>
</tr>
</tbody>
</table>

The pattern navigation, along with aggregate functions, can be qualified with the FINAL and RUNNING semantics keywords. These are effectively a windowing clause within the pattern, defining if the action relates to the whole pattern, or from the start of the pattern to the current row.