Deep Dive into SQL Pattern Matching with MATCH_RECOGNIZE

Just about everything you need to know to be successful with SQL pattern matching

@BigRedDW
oracle-big-data.blogspot.com
Who Am I

Keith Laker
Senior Principal Product Manager, Analytical SQL
Oracle

Twitter: @BigRedDW
Blog: http://oracle-big-data.blogspot.com
Purpose of this presentation is to......

Make you successful with 12c pattern matching
Agenda

1. SQL Pattern Matching – quick recap
2. Using built-in measures to understand your pattern
3. Greedy vs. reluctant quantifiers
4. Unmatched rows vs. empty matches
5. SKIP to where exactly?
6. ALL ROWS vs. ONE ROW
7. Preparing to run a query
8. Predicates: how, where and when
9. Minimizing number of sorts
10. Understanding state machines
11. State machines and the Optimizer
12. What is backtracking?
13. Recommendations for testing
14. Summary
Background

Quick recap
My sample TICKER data set
My sample TICKER data set

• **The ticker sample data set** is for download from our GitHub repository – Analytical SQL

• Or you can get an immediate test drive by using our new [LiveSQL](https://www.oracle.com) service
Tutorials on livesql.us.oracle.com For Pattern Matching
Using MATCH_RECOGNIZE for Pattern Matching

Quick overview of MATCH_RECOGNIZE syntax
Finding V shaped patterns in a ticker stream data set

Using new 12c MATCH_RECOGNIZE to search for patterns

```sql
SELECT symbol, tstamp, price,
       first_down, first_price, last_up, last_price
FROM ticker MATCH_RECOGNIZE (  
   PARTITION BY symbol ORDER BY tstamp
   MEASURES FIRST(str.tstamp) AS first_down,
       FIRST(str.price) as first_price,
       FINAL LAST(up.tstamp) AS last_up,
       FINAL LAST(up.price) as last_price
   ALL ROWS PER MATCH
   PATTERN (strt down+ up+)
   DEFINE
       down AS (price <= PREV(price)),
       up AS (price >= PREV(price))
)
ORDER BY symbol, tstamp;
```

New MATCH_RECOGNIZE clause with table TICKER as input
Overview of 12c MATCH_RECOGNIZE syntax

Partitioning and ordering your data

```
SELECT symbol, tstamp, price,
    first_down, first_price, last_up, last_price
FROM ticker MATCH_RECOGNIZE (  
    PARTITION BY symbol ORDER BY tstamp  
    MEASURES FIRST(strt.tstamp) AS first_down,
           FIRST(strt.price) as first_price,
           FINAL LAST(up.tstamp)  AS last_up,
           FINAL LAST(up.price) as last_price
    ALL ROWS PER MATCH
    PATTERN (strt down+ up+)
    DEFINE
        down AS (price <= PREV(price)),
        up   AS (price >= PREV(price))
)
ORDER BY symbol, tstamp;
```
Overview of 12c MATCH_RECOGNIZE syntax

Describing your pattern

```sql
SELECT symbol, tstamp, price,
    first_down, first_price, last_up, last_price
FROM ticker MATCH_RECOGNIZE (  
    PARTITION BY symbol ORDER BY tstamp  
    MEASURES FIRST(strt.tstamp) AS first_down,  
        FIRST(strt.price) AS first_price,  
        FINAL LAST(up.tstamp) AS last_up,  
        FINAL LAST(up.price) AS last_price
    ALL ROWS PER MATCH
    PATTERN (strt down+ up+)
    DEFINE
        down AS (price <= PREV(price)),
        up AS (price >= PREV(price))
)
ORDER BY symbol, tstamp;
```

Ordered and partitioned stream of rows

Variable names and operators
Overview of 12c MATCH_RECOGNIZE syntax

Defining your pattern

SELECT symbol, tstamp, price,
    first_down, first_price, last_up, last_price
FROM ticker MATCH_RECOGNIZE (  
    PARTITION BY symbol ORDER BY tstamp  
    MEASURES FIRST(strt.tstamp) AS first_down,  
        FIRST(strt.price) as first_price,  
        FINAL LAST(up.tstamp) AS last_up,  
        FINAL LAST(up.price) as last_price  
ALL ROWS PER MATCH  
PATTERN (strt down+ up+)  
DEFINE  
    down AS (price <= PREV(price)),  
    up AS (price >= PREV(price))  
)  
ORDER BY symbol, tstamp;
Overview of 12c MATCH_RECOGNIZE syntax

Defining values to return to SELECT statement

```
SELECT symbol, tstamp, price,
    first_down, first_price, last_up, last_price
FROM ticker MATCH_RECOGNIZE (PARTITION BY symbol ORDER BY tstamp

    MEASURES FIRST(strt.tstamp) AS first_down,
        FIRST(strt.price) as first_price,
        FINAL LAST(up.tstamp) AS last_up,
        FINAL LAST(up.price) as last_price

    ALL ROWS PER MATCH

    PATTERN (strt down+ up+)

    DEFINE
down AS (price <= PREV(price)),
    up AS (price >= PREV(price))
)
ORDER BY symbol, tstamp;
```
Overview of 12c MATCH_RECOGNIZE syntax

Determining how much detail to return to SELECT statement

```
SELECT symbol, tstamp, price,
       first_down, first_price, last_up, last_price
FROM ticker MATCH_RECOGNIZE (  
    PARTITION BY symbol ORDER BY tstamp  
    MEASURES FIRST(strt.tstamp) AS first_down,  
       FIRST(strt.price) as first_price,  
       FINAL LAST(up.tstamp) AS last_up,  
       FINAL LAST(up.price) as last_price  
    ALL ROWS PER MATCH  
    PATTERN (strt down+ up+)  
    DEFINE  
       down AS (price <= PREV(price)),  
       up AS (price >= PREV(price))  
)
ORDER BY symbol, tstamp;
```
Consistent results

• ORDER BY clause is optional in syntax (and ANSI proposal)

• My data is already sorted so I don’t need ORDER BY - correct?
  – Tempting to ignore ORDER BY clause and assume data will be correctly ordered
  – Without ORDER BY, consistent results are not guaranteed!

• **Always** include ORDER BY clause
  – If order of two rows in a partition is not determined by ORDER BY results (non-unique order by key), the result will be non-deterministic
  – If you have non unique order by keys within partition, consider adding additional order by columns to make order by unique and deterministic
  – If Oracle can suppress the order by then it will do so!
Most queries will use **ONE ROW PER MATCH** syntax....

```sql
SELECT symbol, start_tstamp, end_tstamp, match_num
FROM ticker
MATCH_RECOGNIZE (  
    PARTITION BY symbol ORDER BY tstamp
    MEASURES FIRST(down.tstamp) AS start_tstamp,
        LAST(up.tstamp) AS end_tstamp,
        match_number() AS match_num
    ONE ROW PER MATCH
    PATTERN (STRT DOWN+ UP+)
    DEFINE
        DOWN AS (price <= PREV(price)),
        UP AS (price >= PREV(price))
    )
WHERE symbol = 'OSCORP';
```
...which produces a summary report

Not very useful for showing how your pattern is being matched as rows are processed. How do you debug a pattern matching query?

Allows use of built-in measures: MATCH_NUMBER() and CLASSIFIER()
Other queries will use **ALL ROWS PER MATCH** syntax....

```sql
SELECT symbol, tstamp, price, start_tstamp, end_tstamp, match_num, classifier
FROM ticker
MATCH_RECOGNIZE (
    PARTITION BY symbol ORDER BY tstamp
    MEASURES FIRST(down.tstamp) AS start_tstamp,
        LAST(up.tstamp) AS end_tstamp,
        MATCH_NUMBER() AS match_num,
        CLASSIFIER() AS classifier
    ALL ROWS PER MATCH WITH UNMATCHED ROWS
    PATTERN (DOWN+ UP+)
    DEFINE
        DOWN AS (price <= PREV(price)),
        UP AS (price >= PREV(price))
)
WHERE symbol = 'GLOBEX';
```
...which produces a more detailed report

More useful for showing how your pattern is being matched so perfect for debugging a pattern matching query.

Allows additional measures such as CLASSIFIER() and MATCH_NUMBER() to provide more feedback.
Built-in Measures

Using built-in debugging tools to help you understand the pattern matching process

MATCH_NUMBER()
CLASSIFIER()
Two key built-in measures to help with debugging

1. MATCH_NUMBER()
   - Returns an integer to show which rows are members of which match
   - Assigns the same number to each row of a specific match
   - For instance, all the rows in the first match found in a row pattern partition are assigned the match number value of 1
   - Note that match numbering starts over again at 1 in each row pattern partition

2. CLASSIFIER()
   - Shows which rows map to which variable
Example code using two built-in measures

Finding V shaped patterns in a ticker stream

```
SELECT symbol, tstamp, price, mn, pattern,
   first_down, first_price, last_up, last_price
FROM ticker MATCH_RECOGNIZE (   
   PARTITION BY symbol ORDER BY tstamp
   MEASURES MATCH_NUMBER() AS mn,
   CLASSIFIER() as pattern,
   FIRST(strt.tstamp) AS first_down,
   FIRST(strt.price) as first_price,
   FINAL LAST(up.tstamp) AS last_up,
   FINAL LAST(up.price) as last_price
   ALL ROWS PER MATCH
   PATTERN (strt down+ up+)
   DEFINE
      down AS (price <= PREV(price)),
      up AS (price >= PREV(price))
)
ORDER BY symbol, tstamp;
```
1. MATCH_NUMBER()

- MATCH_NUMBER assigns the same number to each row of a specific match
  - First match of complete pattern found in a partition assigned match_number() value of 1
  - Next match gets value of 2, etc.

- Note that match numbering starts over again at 1 in each row pattern partition
2. CLASSIFIER()

• CLASSIFIER() shows which rows map to which variable: **STRT**, **DOWN** or **UP**

• In this example,
  – rows 1, 7, 11, 15 map to variable **STRT**
  – Rows 2, 8, 12, 16 map to variable **DOWN**
  – remaining rows map to variable **UP**
What’s the impact of using ONE ROW PER MATCH?

```sql
SELECT symbol, tstamp, price, mn, pat, first_down, first_price, last_up, last_price
FROM ticker MATCH_RECOGNIZE ( PARTITION BY symbol ORDER BY tstamp
MEASURES price as price,
  tstamp as tstamp,
  MATCH_NUMBER() AS mn,
  CLASSIFIER() as pat,
  FIRST(strt.tstamp) AS first_down,
  FIRST(strt.price) as first_price,
  FINAL LAST(up.tstamp) AS last_up,
  FINAL LAST(up.price) as last_price
ONE ROW PER MATCH
PATTERN (strt down+ up+)
DEFINE
  down AS (price <= PREV(price)),
  up AS (price >= PREV(price))
)
ORDER BY symbol, tstamp;
```
What’s the impact of using ONE ROW PER MATCH?

Note: CLASSIFER() can be used with **ONE ROW PER MATCH**
But only last pattern variable is returned (in this case **UP**) - so not really very useful
Using aggregates with CLASSIFIER()

SELECT symbol, tstamp, price, mn, pat, first_down, first_price, last_up, last_price
FROM ticker MATCH_RECOGNIZE (PARTITION BY symbol ORDER BY tstamp
MEASURES price as price,
tstamp as tstamp,
MATCH_NUMBER() AS mn,
FIRST(CLASSIFIER()) as f_pat,
LAST(CLASSIFIER()) as l_pat
FIRST(strt.tstamp) AS first_down,
FIRST(strt.price) as first_price,
FINAL LAST(up.tstamp) AS last_up,
FINAL LAST(up.price) as last_price
ONE ROW PER MATCH
PATTERN (strt down+ up+)
DEFINE
down AS (price <= PREV(price)),
up AS (price >= PREV(price))
)
ORDER BY symbol, tstamp;
Using aggregates with CLASSIFIER()

ORA-62507: illegal use of MATCH_NUMBER or CLASSIFIER in MATCH_RECOGNIZE clause

62507. 00000 - "illegal use of MATCH_NUMBER or CLASSIFIER in MATCH_RECOGNIZE clause"
*Cause: An attempt was made to use MATCH_NUMBER or CLASSIFIER in the MATCH_RECOGNIZE clause. CLASSIFIER and MATCH_NUMBER can only be used in the MEASURES clause. CLASSIFIER can only be used if the query is ALL ROWS PER MATCH. CLASSIFIER and MATCH_NUMBER cannot be used inside aggregates/FIRST/LAST/PREV/NEXT.

*Action: Modify the query and retry the operation.
Greedy Quantifiers
Using greedy and reluctant quantifiers in your pattern definition
Defining PATTERNs

• **PATTERN** component is used to specify regular expressions

• *Regular expression* is built from variable names and operators
  – Operators can be concatenation, grouping, alternation, permutes, quantifiers, ...
  – A large library of built-on quantifiers is available
  – Regular expressions are amazingly powerful and deeply expressive

• What is a regular expression?
  – *a regular expression (sometimes called a rational expression) is a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace" - like operations*
What is a regular expression?

- Regular expressions used to specify a set of strings (tokens and quantifiers) required for a particular purpose
- Quantifier after a token or group specifies how often that preceding element is allowed to occur
- Most common quantifiers are:
  - Question mark, indicates zero or one match
  - Asterisk, indicates need for zero or more matches
  - Plus sign, indicates need for one or more matches

- Oracle's regular expressions are slightly different
  - Row pattern variables are defined by Boolean conditions rather than characters or sets of characters
Quantifiers used in PATTERN clause

- POSIX basic and extended quantifiers:
  
  - `*` 0 or more matches
  - `+` 1 or more matches
  - `?` 0 or 1 match
  - `{n}` exactly n matches
  - `{n,}` n or more matches
  - `{n, m}` at least n but not more than m (inclusive) matches
  - `{, m}` at most m (inclusive) matches
How to use quantifiers

• The following are examples of using quantifiers:
  – A? matches 0 or 1 iteration of variable A
  – A* matches 0 or more iterations of variable A
  – A+ matches 1 or more iterations of variable A
  – A{3,6} matches 3 to 6 iterations of variable A
  – A{,4} matches 0 to 4 iterations of variable A

• A is defined in the DEFINE component of the MATCH_RECOGNIZE clause
  – For example:  A AS (price <= PREV(price))
Greedy and reluctant quantifiers

• Pattern quantifiers are referred to as **greedy**
  – Attempt to match as many instances as possible of the regular expression on which they are applied

• **Reluctant** quantifiers use a question mark ? as additional suffix
  – Attempt to match as few instances as possible of the regular expression on which they are applied

• Convert greedy to reluctant quantifier by adding additional “?”
  – Examples: ?? or *? or +? or {n, m }?
Example – using greedy quantifiers

Finding V shaped patterns in a ticker stream using plus-sign greedy quantifier

SELECT symbol, tstamp, price, mn, pattern,
    first_down, first_price, last_up, last_price
FROM ticker MATCH_RECOGNIZE (   
    PARTITION BY symbol ORDER BY tstamp
    MEASURES MATCH_NUMBER() AS mn,
        CLASSIFIER() as pattern,
        FIRST(strt.tstamp) AS first_down,
        FIRST(strt.price) as first_price,
        LAST(up.tstamp) AS last_up,
        LAST(up.price) as last_price
    ALL ROWS PER MATCH
    PATTERN (strt down+ up+)
    DEFINE
        down AS (price <= PREV(price)),
        up AS (price >= PREV(price))
) WHERE symbol = 'ACME'
ORDER BY symbol, tstamp;
Using greedy quantifiers

Matching to variable DOWN takes precedence over UP on row 13

Conflicts:
- horizontal area could be mapped to DOWN or UP

Result:
- Greedy DOWN matches as many instances possible to DOWN before matching to UP
Example - using a reluctant quantifier

Using question-mark to indicate reluctant quantifier

SELECT symbol, tstamp, price, mn, pattern, first_down, first_price, last_up, last_price
FROM ticker MATCH_RECOGNIZE (PARTITION BY symbol ORDER BY tstamp
MEASURES MATCH_NUMBER() AS mn,
    CLASSIFIER() as pattern,
    FIRST(strt.tstamp) AS first_down,
    FIRST(strt.price) as first_price,
    LAST(up.tstamp)  AS last_up,
    LAST(up.price) as last_price

ALL ROWS PER MATCH
PATTERN (strt down+? up+)
DEFINE
down AS (price <= PREV(price)),
up AS (price >= PREV(price))
)
WHERE symbol = 'ACME'
ORDER BY symbol, tstamp;
Using a reluctant quantifiers

Matching to variable UP takes precedence over DOWN on row 13

Records matches to UP before considering reluctant DOWN again after having found one match of DOWN already (pattern satisfied)
EMPTY vs. UNMATCHED ROWS
What is the difference between an unmatched row and an empty match?
Empty Matches vs. Unmatched Rows

Two output options for MATCH_RECOGNIZE

• **ONE ROW PER MATCH**: each match produces one summary row. This is the default.

• **ALL ROWS PER MATCH**: a match spanning multiple rows will produce one output row for each row in the match
  – ALL ROWS PER MATCH **SHOW EMPTY MATCHES** <- note that this is the default
  – ALL ROWS PER MATCH **OMIT EMPTY MATCHES**
  – ALL ROWS PER MATCH **WITH UNMATCHED ROWS**
Empty Matches vs. Unmatched Rows

How do you get an “empty match”? 

• Determined by the type of quantifier used as part of the pattern definition
  – Currently possible to get empty matches without quantifiers
    • using a pattern like PATTERN ( ^ | A)
  – Possible to get unmatched rows without using quantifiers

• By changing the quantifier it is possible to produce the similar result using both sets of keywords:
  – ALL ROWS PER MATCH SHOW EMPTY MATCHES <- note that this is the default
  – ALL ROWS PER MATCH WITH UNMATCHED ROWS
Empty Matches

SELECT . . .
FROM ticker
MATCH_RECOGNIZE (PARTITION BY symbol ORDER BY tstamp
MEASURES FIRST(down.tstamp),
LAST(down.tstamp),
match_number() AS match_num,
classifier() AS classifier
ALL ROWS PER MATCH SHOW EMPTY MATCHES
PATTERN (DOWN*)
DEFINE
  DOWN AS (price <= PREV(price))
WHERE symbol = 'GLOBEX';

Empty matches: classifier is NULL even though match_number() returns a value
What is an empty match?

• Empty match is a row that does not map explicitly to a pattern variable
  – in previous example – **DOWN**

• Usually result of using specific quantifier: * (asterisk).
  – Given that **DOWN** variable can be matched zero or more times there is opportunity for an empty match to occur.

• Empty match does in fact have a starting row
  – it is assigned a sequential match number, based on the ordinal position of its starting row
Empty Matches and MATCH_NUMBER

```sql
SELECT . . .
FROM ticker
MATCH_RECOGNIZE (PARTITION BY symbol ORDER BY tstamp
  MEASURES FIRST(down.tstamp),
  LAST(down.ts)
  match_number,
  classifier()
ALL ROWS PER MATCH OMIT EMPTY MATCHES
PATTERN (DOWN*)
DEFINE
  DOWN AS (price <= PREV(price))
WHERE symbol = 'GLOBEX'
```

Note that match_number is not contiguous
Impact of Empty Matches on Analytics

```sql
SELECT *
FROM ticker
MATCH_RECOGNIZE (
PARTITION BY symbol ORDER BY tstamp
MEASURES FIRST(down.tstamp) AS start_time,
LAST(down.tstamp) AS end_time,
COUNT(*) AS counter_all,
avg(down.price) as avg_price,
MATCH_NUMBER() as mn,
CLASSIFIER() as pat
ALL ROWS PER MATCH SHOW EMPTY MATCH PATTERN (DOWN*)
DEFINE DOWN AS (price <= PREV(price))
WHERE symbol = 'GLOBEX';
```

Note that COUNT and AVG only function where CLASSIFIER returns a value.
Summary of Empty Matches

Three mains points to remember when your pattern permits this type of matching:

1. Value of MATCH_NUMBER() is sequential match number of the empty match
2. Any COUNT or similar function (MAX, MIN, AVG etc.) will return 0
3. Any other aggregate, row pattern navigation operation, or ordinary row pattern column reference is null

• The default is always to return empty matches
  – Determine from start how you want to manage these rows:
    • include them (SHOW EMPTY MATCHES)
    • exclude them (OMIT EMPTY MATCHES).
  – Be careful if you are using MATCH_NUMBER() within the DEFINE section as part of a formula because empty matches increment the MATCH_NUMBER() counter.
Viewing Unmatched Rows

• Always useful to view the complete result set
  – at least when you are running your code against test data sets.

• Getting all the input rows into your output is relatively easy because you just need to include the phrase:
  – ALL ROWS PER MATCH WITH UNMATCHED ROWS.

• Other than for testing purposes I can’t think of a good use case for using this in production so make sure you check your code before submitting your production-ready code to your DBA.
Empty Matches and Unmatched Rows In Same Query?

SELECT symbol, tstamp, price, mnm, nmr, cls
FROM ticker MATCH_RECOGNIZE(
  PARTITION BY symbol
  ORDER BY tstamp
  MEASURES match_number() AS mnm,
          count(*) AS nmr,
          classifier() AS cls
  ALL ROWS PER MATCH SHOW EMPTY MATCHES
  PATTERN ((^A*)|A+)
  DEFINE A AS price > 11)
WHERE symbol = 'GLOBEX'
ORDER BY 1, 2;
Empty Matches and Unmatched Rows In Same Query?

```sql
SELECT symbol, tstamp, price, mnm, nmr, cls
FROM ticker MATCH_RECOGNIZE(
    PARTITION BY symbol
    ORDER BY tstamp
    MEASURES match_number() AS mnm,
            count(*) AS nmr,
            classifier() AS cls
    ALL ROWS PER MATCH OMIT EMPTY MATCHES
    PATTERN (\(^A*\)|A+)
    DEFINE A AS price > 11)
WHERE symbol = 'GLOBEX'
ORDER BY 1, 2;
```
Empty Matches and Unmatched Rows

SELECT symbol, tstamp, price, mnm, nmr, cls
FROM ticker MATCH_RECOGNIZE(
    PARTITION BY symbol
    ORDER BY tstamp
    MEASURES match_number() AS mnm,
    count(*) AS nmr,
    classifier() AS cls
    ALL ROWS PER MATCH WITH UNMATCHED ROWS
    PATTERN ((^A*)|A+)
    DEFINE A AS price > 11)
WHERE symbol = 'GLOBEX'
ORDER BY 1, 2;
AFTER MATCH SKIP TO. . .

Where to start searching for the next pattern . . . there are so many places to choose from!
Skipping - basic syntax

- **AFTER MATCH SKIP TO NEXT ROW**
  - Resume pattern matching at the row after the first row of the current match.

- **AFTER MATCH SKIP PAST LAST ROW [DEFAULT]**
  - Resume pattern matching at the next row after the last row of the current match.

- **AFTER MATCH SKIP TO FIRST `pattern_variable`**
  - Resume pattern matching at the first row that is mapped to the pattern variable.

- **AFTER MATCH SKIP TO LAST `pattern_variable`**
  - Resume pattern matching at the last row that is mapped to the pattern variable.

- **AFTER MATCH SKIP TO `pattern_variable`**
  - The same as AFTER MATCH SKIP TO LAST `pattern_variable`. 
### Match #1 Complete: Rows 5 - 10

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>TSTAMP</th>
<th>PRICE</th>
<th>MN</th>
<th>PAT</th>
<th>FIRST_DOWN</th>
<th>FIRST_PRICE</th>
<th>LAST_UP</th>
<th>LAST_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>05-APR-11</td>
<td>25</td>
<td></td>
<td></td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>06-APR-11</td>
<td>12</td>
<td></td>
<td></td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>07-APR-11</td>
<td>15</td>
<td></td>
<td></td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>08-APR-11</td>
<td>20</td>
<td></td>
<td></td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>09-APR-11</td>
<td>24</td>
<td></td>
<td></td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>11-APR-11</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACME</td>
<td>12-APR-11</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACME</td>
<td>13-APR-11</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACME</td>
<td>14-APR-11</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACME</td>
<td>15-APR-11</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACME</td>
<td>16-APR-11</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACME</td>
<td>17-APR-11</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACME</td>
<td>18-APR-11</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NEXT ROW: Restart Searching For Next Match At Row 6

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>TSTAMP</th>
<th>PRICE</th>
<th>MN</th>
<th>PAT</th>
<th>FIRST_DOWN</th>
<th>FIRST_PRICE</th>
<th>LAST_UP</th>
<th>LAST_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>05-APR-11</td>
<td>25</td>
<td>1</td>
<td>STKT</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>06-APR-11</td>
<td>12</td>
<td>6</td>
<td>1</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>07-APR-11</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>08-APR-11</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>09-APR-11</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>10-APR-11</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>11-APR-11</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>12-APR-11</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>14-APR-11</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>16-APR-11</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>17-APR-11</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>18-APR-11</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>05-APR-11</td>
<td>25 10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>
### LAST ROW: Restart Searching For Next Match At Row 11

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>TSTAMP</th>
<th>PRICE</th>
<th>MN</th>
<th>PAT</th>
<th>FIRST_DOWN</th>
<th>FIRST_PRICE</th>
<th>LAST_UP</th>
<th>LAST_PRICE</th>
</tr>
</thead>
</table>

**Default processing**
**FIRST** `pattern_variable`: Restart Searching For Next Match At Row 6

- **AFTER MATCH SKIP TO NEXT ROW**
- **AFTER MATCH SKIP PAST LAST ROW**
- **AFTER MATCH SKIP TO FIRST DOWN**

Scanning for next match

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>TSTAMP</th>
<th>PRICE</th>
<th>MIN</th>
<th>PAT</th>
<th>FIRST_DOWN</th>
<th>FIRST_PRICE</th>
<th>LAST_UP</th>
<th>LAST_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>05-APR-11</td>
<td>25</td>
<td>1</td>
<td>STNT</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>06-APR-11</td>
<td>12</td>
<td>1</td>
<td>DOWN</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>07-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>08-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>09-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>10-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>11-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>12-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>13-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>14-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>15-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>16-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>17-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>18-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25-10-APR-11</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>
LAST _pattern_variable_: Restart Searching For Next Match At Row 10

- AFTER MATCH SKIP TO NEXT ROW
- AFTER MATCH SKIP PAST LAST ROW
- AFTER MATCH SKIP TO FIRST DOWN
- AFTER MATCH SKIP TO LAST UP

Scanning for next match
**Pattern variable**: Restart Searching For Next Match At Row 10

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>TSTAMP</th>
<th>PRICE</th>
<th>MN</th>
<th>PAT</th>
<th>FIRST_DOWN</th>
<th>FIRST_PRICE</th>
<th>LAST_UP</th>
<th>LAST_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>05-APR-11</td>
<td>25</td>
<td>1</td>
<td>STNT</td>
<td>05-APR-11</td>
<td>25</td>
<td>10-APR-11</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>06-APR-11</td>
<td>12</td>
<td>1</td>
<td>DOWN</td>
<td>05-APR-11</td>
<td>25</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>07-APR-11</td>
<td>15</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25</td>
<td>10-APR-11</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>08-APR-11</td>
<td>1</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25</td>
<td>10-APR-11</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>09-APR-11</td>
<td>15</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25</td>
<td>10-APR-11</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>10-APR-11</td>
<td>15</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25</td>
<td>10-APR-11</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>11-APR-11</td>
<td>15</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25</td>
<td>10-APR-11</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>12-APR-11</td>
<td>15</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25</td>
<td>10-APR-11</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>13-APR-11</td>
<td>15</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>25</td>
<td>10-APR-11</td>
<td>25</td>
</tr>
</tbody>
</table>

- After match skip to next row
- After match skip past last row
- After match skip to first down
- After match skip to last up
- After match skip to up
Looking for shapes and controlling skipping

Multiple Overlapping W-Shapes in the Ticker Data
Looking for multiple W-shapes using AFTER MATCH SKIP...

Finding only 1 W-shape

```sql
SELECT *
FROM Ticker MATCH_RECOGNIZE (
  PARTITION BY symbol ORDER BY tstamp
  MEASURES STRT.tstamp AS start_w,
  LAST(z.tstamp) AS end_w
  ONE ROW PER MATCH
  AFTER MATCH SKIP PAST LAST ROW
  PATTERN (STRT x+ y+ w+ z+)
  DEFINE
    x AS x.price <= PREV(x.price),
    y AS y.price >= PREV(y.price),
    w AS w.price <= PREV(w.price),
    z AS z.price >= PREV(z.price)
  ) MR
WHERE symbol='ACME'
ORDER BY symbol, MR.start_w;
```
Looking for multiple W-shapes using AFTER MATCH SKIP...

Finding only 1 W-shape

Resume searching for next pattern here, means we can’t find another complete W-shape
Looking for multiple W-shapes using AFTER MATCH SKIP...

Finding 2 W-shapes

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (PARTITION BY symbol ORDER BY tstamp
  MEASURES STRT.tstamp AS start_w,
          LAST(z.tstamp) AS end_w
ONE ROW PER MATCH
AFTER MATCH SKIP TO LAST Y
PATTERN (STRT x+ y+ w+ z+)
DEFINE
  x AS x.price <= PREV(x.price),
  y AS y.price >= PREV(y.price),
  w AS w.price <= PREV(w.price),
  z AS z.price >= PREV(z.price)
) MR
WHERE symbol='ACME'
ORDER BY symbol, MR.start_w;
```
Overlapping W-Shapes – rows processed multiple times

Finding 2 w-shapes

Rows 10-Apr-11 to 14-Apr-11 are processed again during search for pattern #2
Looking for shapes and controlling skipping
Multiple Overlapping W-Shapes in the Ticker Data
Looking for multiple W-shapes using AFTER MATCH SKIP...

Finding 3 W-shapes

```sql
SELECT *
FROM Ticker MATCH_RECOGNIZE (  
    PARTITION BY symbol ORDER BY tstamp  
    MEASURES STRT.tstamp AS start_w,  
        LAST(z.tstamp) AS end_w  
    ONE ROW PER MATCH  
    AFTER MATCH SKIP TO NEXT ROW  
    PATTERN (STRT x+ y+ w+ z+)  
    DEFINE  
        x AS x.price <= PREV(x.price),  
        y AS y.price >= PREV(y.price),  
        w AS w.price <= PREV(w.price),  
        z AS z.price >= PREV(z.price)  
    ) MR  
WHERE symbol='ACME'
ORDER BY symbol, mr.start_w;
```
Looking for multiple W-shapes using AFTER MATCH SKIP...

Finding 3 W-shapes

AFTER MATCH SKIP TO NEXT ROW
Resume searching for next pattern at the row after the first row of the current match. Therefore, we find 3 complete W-shapes
ONE ROW vs. ALL ROWS. . .

Returning only the rows you are interested in. . .with empty matches vs. omit empty matches vs. unmatched rows vs. !
• ...content being prepared
Preparing to run a query
How does the optimizer prepare to run a query containing a MATCH_RECOGNIZE clause?
Optimizing MATCH_RECOGNIZE

Costing

Cost of the MATCH_RECOGNIZE evaluation is based on:

1. Cost of sorting input data
   - execution of state machine requires data is partitioned and ordered according to the partition by and order by keys
   - If incoming data is not correctly ordered then sorting of the data is needed and sorting cost is added

2. Cost of executing the state machine
   - this cost is affected by many factors such as pattern regular expression (and whether finite state machine is built), predicates defining variables, match options, AFTER MATCH SKIP to option.
Optimizing MATCH_RECOGNIZE

Sort elimination

• Optimizer attempts to eliminate sort requested by a query by selecting an access method which returns ordered data
  – for example index access or sort-merge-join access.

• Optimizer compares:
  1. cost of the optimal plan it found so far including the sort cost and
  2. cost of plan with order preserving access method
  – Obviously it selects the cheapest one!

• If sort can be eliminated, optimizer does not add its cost to the cost of query block.
Optimizing MATCH_RECOGNIZE

Calculating Cardinality

• Calculation affected by many factors:
  – pattern regular expression
  – variable defining conditions
  – rows per match option + after match skip to option

• Getting an accurate estimate is not feasible.
  – ALL ROWS PER MATCH - cardinality estimate will be set to that of row pattern input table
  – ONE ROW PER MATCH - cardinality estimate will be set to estimate of total number of distinct PARTITION BY values out row pattern input table
How are the standard query transformations implemented?

• There are a number of query transformations that are affected by MATCH_RECOGIZE such as:

  • View Merging
  • Un-nesting
  • Pushing join predicates
  • Pushing predicates
  • Predicate move around
  • Predicate pull up
  • Group pruning
  • Group by placement
  • Materialized view rewrite
How are the standard query transformations implemented?

• View Merging
  – Parser expands a MATCH_RECOGNIZE table expression into an inline view query block
    • MATCH_RECOGNIZE e view query block involves pattern recognition evaluation only
  – View merging is disabled for MATCH_RECOGNIZE view query block
  – If MATCH_RECOGNIZE is applied on a view, then merging of input view with its outer query block (i.e. MATCH_RECOGNIZE view query block) is also disabled
  – **BUT** note that, if MATCH_RECOGNIZE clause applies on a view, then all the query transformations currently available are still available within view

• Un-nesting
  – Not done for subqueries containing MATCH_RECOGNIZE clause
How are the standard query transformations implemented?

• Pushing join predicates
  – Similar to window functions, pushing of join predicates into match recognize view (cost-based) will be enabled provided join keys are on partition by keys

• Pushing predicates
  – Predicates will be pushed on the partition by keys and will filter entire partitions

• Predicate Move Around
  – Predicates will be exported from the un-mergeable match recognize view only if they are on the partition by keys
How are the standard query transformations implemented?

• Group Pruning for Grouping Sets
  – not be affected as a query block can not contain grouping

• GROUP BY placement
  – group by placement transformation will be disabled for views using MATCH_RECOGNIZE

• Materialized views rewrite
  – If MV has a MATCH_RECOGNIZE CLAUSE then only full or partial text match rewrite will be supported
Materialized Views - some restrictions

• Conventional fast refresh will be disabled

• PCT refresh is ONLY feasible and supported IF:
  – MV and base tables are partitioned and the partitioning columns are a subset of the partition-by keys of MATCH_RECOGNIZE clause
New Keywords in Explain Plans
Overview
New Key Words in Explain Plans

• Four new key words relating to pattern matching that will appear in your explain plan:

1. MATCH RECOGNIZE
2. SORT
3. BUFFER
4. DETERMINISTIC FINITE AUTO

• Important keyword in terms of performance – deterministic finite auto state machines deliver faster processing!
PREDICATES

When, where, how and why are they applied when using MATCH_RECOGNIZE?
When exactly is the WHERE clause processed?

Applying predicates to PARTITION BY column

```sql
SELECT symbol, first_x, last_z
FROM ticker
MATCH_RECOGNIZE (  
    PARTITION BY symbol ORDER BY tstamp  
    MEASURES FIRST(x.tstamp) AS first_x,  
    LAST(z.tstamp) AS last_z  
    ONE ROW PER MATCH  
    PATTERN (X+ Y+ W+ Z+)  
    DEFINE X AS (price < PREV(price)),  
                Y AS (price > PREV(price)),  
                W AS (price < PREV(price)),  
                Z AS (price > PREV(price)  
                      AND z.tstamp - FIRST(x.tstamp) <= 7 ))  
WHERE symbol EXISTS ('ACME', 'OSCORP');
```
Applying predicates to ORDER BY column

```
SELECT symbol, first_x, last_z
FROM ticker
MATCH_RECOGNIZE (PARTITION BY symbol ORDER BY tstamp
MEASURES FIRST(x.tstamp) AS first_x,
LAST(z.tstamp) AS last_z
    tstamp AS tstamp
ONE ROW PER MATCH
PATTERN (X+ Y+ W+ Z+)
DEFINE X AS (price < PREV(price)),
    Y AS (price > PREV(price)),
    W AS (price < PREV(price)),
    Z AS (price > PREV(price)
     AND z.tstamp - FIRST(x.tstamp) <= 7 ))
WHERE symbol IN ('ACME', 'OSCORP')
AND tstamp BETWEEN '01-Apr-11' AND '09-Apr-11';
```
Applying predicates to other columns

```sql
SELECT symbol, first_x, last_z
FROM ticker
MATCH_RECOGNIZE (PARTITION BY symbol ORDER BY tstamp
MEASURES FIRST(x.tstamp) AS first_x,
    LAST(z.tstamp) AS last_z,
    tstamp AS tstamp,
    price as price
ONE ROW PER MATCH
PATTERN (X+ Y+ W+ Z+)
DEFINE X AS (price < PREV(price)),
    Y AS (price > PREV(price)),
    W AS (price < PREV(price)),
    Z AS (price > PREV(price))
    AND z.tstamp - FIRST(x.tstamp) <= 7 )
WHERE symbol IN ('ACME', 'OSCORP')
AND tstamp BETWEEN '01-Apr-11' and '09-Apr-11'
AND price > 0;
```
SORT vs. BUFFER in the explain plan

SELECT symbol, first_x, last_z
FROM ticker
MATCH_RECOGNIZE (PARTITION BY symbol ORDER BY tstamp
MEASURES FIRST(x.tstamp) AS first_x,
    LAST(z.tstamp) AS last_z,
    tstamp AS tstamp,
    price as price
ONE ROW PER MATCH
PATTERN (X+ Y+ W+ Z+)
DEFINE X AS (price < PREV(price)),
    Y AS (price > PREV(price)),
    W AS (price < PREV(price)),
    Z AS (price > PREV(price)
    AND z.tstamp - FIRST(x.tstamp) <= 7 ))
WHERE symbol IN ('ACME', 'OSCORP')
AND tstamp BETWEEN '01-Apr-11' and '09-Apr-11';
SORTING

When does sorting happen and how to minimize number of sorts in your explain plan!
New keywords in explain plans

• Four new key words relating to pattern matching that will appear in your explain plan:

1. MATCH RECOGNIZE
2. SORT
3. BUFFER
4. DETERMINISTIC FINITE AUTO
Sort vs. Buffer keywords in explain plan

• Most example/sample explain plans show **SORT** keyword

But...
  – Applying predicates and an **index** is available that provides the correct sort order
  – No predicates applied but **index** available that provides the correct sort order

• Create an index on the columns symbol and tstamp (PARTITION BY and ORDER BY columns)

```sql
CREATE INDEX ticker_tstamp_idx ON ticker(symbol, tstamp)
```
Sort vs. Buffer keywords in explain plan

```
SELECT symbol, first_x, last_z
FROM ticker
MATCH_RECOGNIZE (
    PARTITION BY symbol ORDER BY tstamp
    MEASURES FIRST(x.tstamp) AS first_x,
        LAST(z.tstamp) AS last_z,
        tstamp AS tstamp,                price as price
    ONE ROW PER MATCH
    PATTERN (X+ Y+ W+ Z+)
    DEFINE X AS (price < PREV(price)),
        Y AS (price > PREV(price)),
        W AS (price < PREV(price)),
        Z AS (price > PREV(price) AND z.tstamp - FIRST(x.tstamp) <= 7 ));
WHERE symbol IN ('ACME', 'OSCORP')
AND tstamp BETWEEN '01-Apr-11' AND '09-Apr-11')
```
Sorts vs. Buffers

- Using predicates to filter on symbol
- SORT keyword within the MATCH_RECOGNIZE line has disappeared
- Index is able to provide a correctly ordered row set into MATCH_RECOGNIZE
- MATCH_RECOGNIZE line is showing the keyword BUFFER
  - Indicates no additional sorting is being applied to data as it flows into matching process
Sort vs. Buffer keywords with no predicates

- No predicates - index will actually be used
- MATCH_RECOGNIZE is smart enough to spot that index is useful and can provide the ordering needed for matching processing
  - Index provides correct sort order and explain plan switches to using BUFFER keyword
Sort vs. Buffer keywords with no predicates

SELECT symbol, first_x, last_z
FROM ticker
MATCH_RECOGNIZE (  
    PARTITION BY symbol ORDER BY tstamp
    MEASURES FIRST(x.tstamp) AS first_x,
    LAST(z.tstamp) AS last_z,
    tstamp AS tstamp,
    price as price
    ONE ROW PER MATCH
    PATTERN (X+ Y+ W+ Z+)
    DEFINE X AS (price < PREV(price)),
                Y AS (price > PREV(price)),
                W AS (price < PREV(price)),
                Z AS (price > PREV(price) AND z.tstamp - FIRST(x.tstamp) <= 7 ));
Sorts vs. Buffers with no predicates
Understanding State Machines

…and why you need to care about them!
MATCH_RECOGNIZE and State Machines

• Compilation phase of MATCH_RECOGNIZE generates a finite state machine

• What is an FSM?

A finite-state machine (FSM) ...is a mathematical model of computation ....it is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time ...changes from one state to another when initiated by a triggering event or condition; this is called a transition.

A particular FSM is defined by a list of its states, and the triggering condition for each transition.

Turnstile State Machine

- Has two states: Locked and Unlocked
- Two events affect its state:
  - Putting a coin in the slot (coin)
  - Pushing the arm (push)
- Locked state, pushing on the arm has no effect
- Putting a coin in shifts the state from Locked to Unlocked
  - Putting additional coins in has no effect;
- Pushing through the arms, giving a push input, shifts the state back to Locked

Images courtesy of wikipedia
MATCH_RECOGNIZE and State Machines

• State machine is constructed using information in PATTERN and DEFINE components:
• State machine represented by a directed graph called a state diagram
  – Each state is represented by a node (circle)
  – Edges (arrows) show transitions from one state to another.
    • Labeled with the event (condition) that triggers transition.
  – Events (conditions) that don't cause a change of state are represented by a circular arrow returning to the original state.
State Machine for our pattern....

- Graph representation of (STRT DOWN+? UP+ FLAT+)

- Note the precedence of UP over DOWN for reluctant quantifier DOWN

Precedence is to read from top ->down
MATCH_RECOGNIZE and the Optimizer

Review of new pattern matching keywords in the explain plan
New keywords in explain plans

• Four new key words relating to pattern matching that will appear in your explain plan:

1. MATCH RECOGNIZE
2. SORT
3. BUFFER
4. DETERMINISTIC FINITE AUTO
MATCH_RECOGNIZE plans based on State Machines

• Compilation phase of MATCH_RECOGNIZE generates a finite state machine

• Details of PATTERN component determine if state machine is:
  1. Deterministic Finite Auto (DFA)
     • Each of its transitions is uniquely determined by its source state and event
     • DFA uses an efficient algorithm without backtracking, runs in linear time
  2. Nondeterministic Finite Auto (NFA)
     • Next state of an NFA depends not only on the current event, but also possibly on an arbitrary number of subsequent events
     • NFA implements back tracking + other optimization techniques
MATCH_RECOGNIZE plans based on State Machines

Explain plan indicates which algorithm is used:
SELECT symbol, start_tstamp, end_tstamp, match_num
FROM Ticker
MATCH_RECOGNIZE
     PARTITION BY symbol ORDER BY tstamp
     MEASURES STRT.tstamp AS start_tstamp,
             LAST(UP.tstamp) AS end_tstamp,
             match_number() AS match_num
     ONE ROW PER MATCH
     AFTER MATCH SKIP TO LAST UP
     PATTERN (STRT DOWN UP)
     DEFINE
             DOWN AS DOWN.price < PREV(DOWN.price),
             UP AS UP.price > PREV(UP.price)
     ) WHERE symbol= ‘ACME’ ;
Built A Deterministic Finite State Machine...

No backtracking and runs in linear time. . .

```
SELECT *
FROM Ticker
MATCH_RECOGNIZE (
    PARTITION BY symbol ORDER BY tstamp
    MEASURES strt.tstamp AS start_tstamp,
        LAST(down.tstamp) AS b_tstamp,
        LAST(up.tstamp) AS e_tstamp
    ONE ROW PER MATCH
    AFTER MATCH SKIP TO LAST UP
    PATTERN (STRT DOWN UP)
    DEFINE
        down AS price < PREV(price),
        up AS price > PREV(price)
) WHERE symbol= 'ACME';
```
Managed to build Deterministic Finite State Machine...

```
SELECT symbol, start_tstamp, end_tstamp, match_num
FROM Ticker
MATCH_RECOGNIZE (PARTITION BY symbol ORDER BY tstamp
MEASURES STRT.tstamp AS start_tstamp,
       LAST(UP.tstamp) AS end_tstamp,
       match_number() AS match_num
ONE ROW PER MATCH
AFTER MATCH SKIP TO LAST UP
PATTERN (STRT DOWN UP*)
DEFINE
  DOWN AS DOWN.price < PREV(DOWN.price),
  UP AS UP.price > PREV(UP.price)
) WHERE symbol= 'ACME';
```
Built A Deterministic Finite State Machine...

• Key attribute of a Deterministic Finite Auto:

  1. Each of its transitions is uniquely determined by its source state and event

      PATTERN (STRT DOWN UP*)
      DEFINE
      DOWN AS DOWN.price < PREV(DOWN.price),
      UP AS UP.price > PREV(UP.price)

• Therefore, the DFA state machine is able to use an efficient algorithm (no backtracking) and runs in linear time
Didn’t manage to build Deterministic Finite State Machine...

```
SELECT symbol, start_tstamp, end_tstamp, match_num
FROM Ticker
MATCH_RECOGNIZE
  PARTITION BY symbol ORDER BY tstamp
  MEASURES STRT.tstamp AS start_tstamp,
       LAST(UP.tstamp) AS end_tstamp,
       match_number() AS match_num
ONE ROW PER MATCH
AFTER MATCH SKIP TO LAST UP
PATTERN (STRT DOWN* UP*)
DEFINE
  DOWN AS DOWN.price < PREV(DOWN.price),
  UP AS UP.price > PREV(UP.price)
WHERE symbol = 'ACME';
```
Could Not Build A Deterministic Finite State Machine...

• Key reason why it is not possible to build Deterministic Finite Auto state machine:

  1. Next state of an NFA depends not only on the current event, but also possibly on an arbitrary number of subsequent events

    \[
    \text{PATTERN (STRT DOWN* UP*)}
    \]
    
    DEFINE DOWN AS DOWN.price < PREV(DOWN.price),
    UP AS UP.price > PREV(UP.price)

• Therefore:

  1. NFA implements back tracking
  2. But other \textit{optimization} techniques implemented to ensure good performance
Non-deterministic state machine

Determinism unknown, backtracking in place, runs in non-linear time.

```sql
SELECT *
FROM Ticker
MATCH_RECOGNIZE (  
    PARTITION BY symbol ORDER BY tstamp  
    MEASURES strt.tstamp AS start_tstamp,  
       LAST(down.tstamp) AS b_tstamp,  
       LAST(up.tstamp) AS e_tstamp  
    ONE ROW PER MATCH  
    AFTER MATCH SKIP TO LAST up  
    PATTERN (strt down* up*)  
    DEFINE  
       down AS price <= PREV(price),  
       up   AS price >= PREV(price)  
) WHERE symbol= 'ACME' ;
```
What is Backtracking
Why should I care about backtracking?
Extending the pattern to test final price vs. initial price

```sql
SELECT symbol, mn, tstamp, pattern, price, first_price, last_price
FROM ticker MATCH_RECOGNIZE ( PARTITION BY symbol ORDER BY tstamp
MEASURES MATCH_NUMBER() AS mn,
    FIRST(strt.tstamp) AS first_x,
    FIRST(strt.price) AS first_price,
    LAST(z.tstamp) AS last_z,
    last(z.price) AS last_price,
    classifier() AS pattern
ALL ROWS PER MATCH WITH UNMATCHED ROWS
PATTERN (STRT X+ Y+ W+ Z+ AVGP)
DEFINE X AS (price < PREV(price)),
    Y AS (price > PREV(price)),
    W AS (price < PREV(price)),
    Z AS (price > PREV(price)),
    AVGP AS (last(z.price) < strt.price*1.5)) ;
```
Comparing results of first and second statement

How are the rows of data now processed through the non-deterministic state machine?

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MN</th>
<th>TSTAMP</th>
<th>PATTERN</th>
<th>PRICE</th>
<th>FIRST_PRICE</th>
<th>LAST_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OSCORP</td>
<td>01-APR-11</td>
<td></td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>OSCORP</td>
<td>02-APR-11</td>
<td>STRT</td>
<td>22</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>OSCORP</td>
<td>03-APR-11</td>
<td>X</td>
<td>19</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>OSCORP</td>
<td>04-APR-11</td>
<td>X</td>
<td>18</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>OSCORP</td>
<td>05-APR-11</td>
<td>X</td>
<td>17</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>

Basic Pattern

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MN</th>
<th>TSTAMP</th>
<th>PATTERN</th>
<th>PRICE</th>
<th>FIRST_PRICE</th>
<th>LAST_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OSCORP</td>
<td>01-APR-11</td>
<td></td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>OSCORP</td>
<td>02-APR-11</td>
<td></td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>OSCORP</td>
<td>03-APR-11</td>
<td>STRT</td>
<td>19</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>OSCORP</td>
<td>04-APR-11</td>
<td>X</td>
<td>18</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>OSCORP</td>
<td>05-APR-11</td>
<td>X</td>
<td>17</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>

Modified pattern with AVGP as final part of pattern
Comparing results of first and second statement

1. ROW 2 is not matched anymore to always-true event STRT
2. STRT variable now matched at row 3.
3. W-pattern still ends at ROW 8
4. ROW 9 is now mapped to variable AVGP

Backtracking in action!
(STRT X+ Y+ W+ Z+ AVGP)

Start 0 \(\text{strt}\) State 1 (strt) x State 2 (x) y State 3 (y) w State 4 (w) z State 5 (z) avgp State 6 (avgp)

<table>
<thead>
<tr>
<th>Current state</th>
<th>Row evaluated</th>
<th>Event</th>
<th>Event met</th>
<th>New state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>strt</td>
<td>Y</td>
<td>1</td>
</tr>
</tbody>
</table>
Pattern has failed to match
Move pointer to next row: Row 2
Start applying state machine to test for pattern
(STRT X+ Y+ W+ Z+ AVGP)

Start

State 1 (strt)

State 2 (x)

State 3 (y)

State 4 (w)

State 5 (z)

State 6 (avgp)

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MN</th>
<th>TSTAMP</th>
<th>PATTERN</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OSCORP</td>
<td>01-APR-11</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>OSCORP</td>
<td>02-APR-11 STRT</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current state</th>
<th>Row evaluated</th>
<th>Event</th>
<th>Event met</th>
<th>New state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>strt</td>
<td>Y</td>
<td>1</td>
</tr>
</tbody>
</table>
(STRT X+ Y+ W+ Z+ AVGP)

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MN</th>
<th>TSTAMP</th>
<th>PATTERN</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSCORP</td>
<td>01-APR-11</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>102-APR-11 STRT</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>103-APR-11 X</td>
<td>19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current state</th>
<th>Row evaluated</th>
<th>Event</th>
<th>Event met</th>
<th>New state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>strt</td>
<td>Y</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>X</td>
<td>Y</td>
<td>2</td>
</tr>
</tbody>
</table>

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.
Current state | Row evaluated | Event | Event met | New state
---|---|---|---|---
1 | 2 | strt | Y | 1
2 | 3 | X | Y | 2
2 | 4 | X | Y | 2
### Current state | Row evaluated | Event | Event met | New state
--- | --- | --- | --- | ---
1 | 2 | strt | Y | 1
2 | 3 | X | Y | 2
2 | 4 | X | Y | 2
2 | 5 | X | Y | 2

**Diagram**:

- **START (strt)**
- **State 1 (strt)**
- **State 2 (x)**
- **State 3 (y)**
- **State 4 (w)**
- **State 5 (z)**
- **State 6 (avgp)**

**Transitions**:
- **Start → State 1**: strt
- **State 1 → State 2**: x
- **State 2 → State 3**: y
- **State 3 → State 4**: w
- **State 4 → State 5**: z
- **State 5 → State 6**: avgp

**Table**:

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MN</th>
<th>TSTAMP</th>
<th>PATTERN</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSCORP</td>
<td>01-APR-11</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>102-APR-11 STRT</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>103-APR-11 X</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>104-APR-11 X</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>105-APR-11 X</td>
<td>17</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(STRT X+ Y+ W+ Z+ AVGP)

Current state  | Row evaluated | Event | Event met | New state
---|--|--|--|--
1 | 2 | strt | Y | 1
2 | 3 | X | Y | 2
2 | 4 | X | Y | 2
2 | 5 | X | Y | 2
2 | 6 | X | N | 2
(STRT X+ Y+ W+ Z+ AVGP)

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MN</th>
<th>TSTAMP</th>
<th>PATTERN</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OSCORP</td>
<td>01-APR-11</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>OSCORP</td>
<td>1 02-APR-11 STRT</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>OSCORP</td>
<td>1 03-APR-11 X</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>OSCORP</td>
<td>1 04-APR-11 X</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>OSCORP</td>
<td>1 05-APR-11 X</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>OSCORP</td>
<td>1 06-APR-11 Y</td>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current state</th>
<th>Row evaluated</th>
<th>Event</th>
<th>Event met</th>
<th>New state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>strt</td>
<td>Y</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>X</td>
<td>Y</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>X</td>
<td>Y</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>X</td>
<td>Y</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>X</td>
<td>N</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>Y</td>
<td>Y</td>
<td>3</td>
</tr>
</tbody>
</table>
### Table

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MN</th>
<th>TSTAMP</th>
<th>PATTERN</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 OSCORP</td>
<td>01-APR-11</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 OSCORP</td>
<td>1.02-APR-11 STRT</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 OSCORP</td>
<td>1.03-APR-11 X</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 OSCORP</td>
<td>1.04-APR-11 X</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 OSCORP</td>
<td>1.05-APR-11 X</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 OSCORP</td>
<td>1.06-APR-11 Y</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 OSCORP</td>
<td>1.07-APR-11 W</td>
<td>17</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Diagram

**Current state**

<table>
<thead>
<tr>
<th>Current state</th>
<th>Row evaluated</th>
<th>Event</th>
<th>Event met</th>
<th>New state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>strt</td>
<td>Y</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>X</td>
<td>Y</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>X</td>
<td>Y</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>X</td>
<td>Y</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>X</td>
<td>N</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>Y</td>
<td>Y</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>Y</td>
<td>N</td>
<td>3</td>
</tr>
</tbody>
</table>
(STRT X+ Y+ W+ Z+ AVGP)

Current state | Row evaluated | Event | Event met | New state
---|---|---|---|---
1 | 2 | STRT | Y | 1
2 | 3 | X | Y | 2
2 | 4 | X | Y | 2
2 | 5 | X | Y | 2
2 | 6 | X | N | 2
2 | 6 | Y | Y | 3
3 | 7 | Y | N | 3
3 | 7 | W | Y | 4
(STRT X+ Y+ W+ Z+ AVGP)

Current state | Row evaluated | Event | Event met | New state
---|---|---|---|---
1 | 2 | strt | Y | 1
2 | 3 | X | Y | 2
2 | 4 | X | Y | 2
2 | 5 | X | Y | 2
2 | 6 | X | N | 2
2 | 6 | Y | Y | 3
3 | 7 | Y | N | 3
3 | 7 | W | Y | 4
4 | 8 | W | N | 4
4 | 8 | Z | Y | 5
(STRT X+ Y+ W+ Z+ AVGP)

Start
State 1 (strt)
State 2 (x)
State 3 (y)
State 4 (w)
State 5 (z)
State 6 (avgp)

Current state | Row evaluated | Event | Event met | New state
--- | --- | --- | --- | ---
1 | 2 | strt | Y | 1
2 | 3 | X | Y | 2
2 | 4 | X | Y | 2
2 | 5 | X | Y | 2
2 | 6 | X | N | 2
2 | 6 | Y | Y | 3
3 | 7 | Y | N | 3
3 | 7 | W | Y | 4
4 | 8 | W | N | 4
4 | 8 | Z | Y | 5
5 | 9 | Z | N | 5
### Current state | Row evaluated | Event | Event met | New state
---|---|---|---|---
1 | 2 | strt | Y | 1
2 | 3 | X | Y | 2
3 | 4 | X | Y | 2
4 | 5 | X | Y | 2
2 | 6 | X | N | 2
2 | 6 | Y | Y | 3
3 | 7 | Y | N | 3
3 | 7 | W | Y | 4
4 | 8 | W | N | 4
4 | 8 | Z | Y | 5
5 | 9 | Z | N | 5
5 | 9 | Avgp (2 – 8) | N | FAIL
(STRT X+ Y+ W+ Z+ AVGP)

Start

State 1 (strt)

State 2 (x)

State 3 (y)

State 4 (w)

State 5 (z)

State 6 (avgp)

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MN</th>
<th>TSTAMP</th>
<th>PATTERN</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OSCORP</td>
<td>1-APR-11</td>
<td>STRT</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>OSCORP</td>
<td>1.02-APR-11</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>OSCORP</td>
<td>1.03-APR-11</td>
<td>X</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>OSCORP</td>
<td>1.04-APR-11</td>
<td>X</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>OSCORP</td>
<td>1.05-APR-11</td>
<td>X</td>
<td>17</td>
</tr>
</tbody>
</table>

Current state | Row evaluated | Event | Event met | New state
---|---|---|---|---|
1 | 2 | strt | Y | 1 |
2 | 3 | X | Y | 2 |
2 | 4 | X | Y | 2 |
2 | 5 | X | Y | 2 |
(STRT X+ Y+ W+ Z+ AVGP)

Current state | Row evaluated | Event | Event met | New state
---|---|---|---|---
1 | 2 | strt | Y | 1
2 | 3 | X | Y | 2
3 | 4 | X | Y | 2
4 | 5 | X | IGNORE | 2
5 | 5 | Y | N | FAIL
BACKTRACKING STARTED

STARTED
(STRT X+ Y+ W+ Z+ AVGP)

Current state | Row evaluated | Event | Event met | New state
---|---|---|---|---
1 | 2 | strt | Y | 1
2 | 3 | X | Y | FAIL

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MN</th>
<th>TSTAMP</th>
<th>PATTERN</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSCORP</td>
<td>01·APR·11</td>
<td></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>OSCORP</td>
<td>102·APR·11·STRT</td>
<td></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>OSCORP</td>
<td>103·APR·11·X</td>
<td></td>
<td></td>
<td>19</td>
</tr>
</tbody>
</table>

BACKTRACKING STARTED
Can’t backtrack any further because there are no more matches for X
**Pattern has failed to match!**
- Move pointer to next row: Row 3
- Start applying state machine to test for pattern
(STRT \ x+ \ y+ \ w+ \ z+ \ AVG\ P)

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MN</th>
<th>TSTAMP</th>
<th>PATTERN</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OSCORP</td>
<td>01-APR-11</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>OSCORP</td>
<td>1.02-APR-11</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>OSCORP</td>
<td>1.03-APR-11</td>
<td>STRT</td>
<td>19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current state</th>
<th>Row evaluated</th>
<th>Event</th>
<th>Event met</th>
<th>New state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>strt</td>
<td>Y</td>
<td>1</td>
</tr>
</tbody>
</table>
Pattern completes: starts Row 3, ends Row 9

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MIN</th>
<th>TSTAMP</th>
<th>PATTERN</th>
<th>PRICE</th>
<th>FIRST_PRICE</th>
<th>LAST_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSCORP</td>
<td>null</td>
<td>01-APR-11 (null)</td>
<td>22</td>
<td>(null)</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>null</td>
<td>02-APR-11 (null)</td>
<td>22</td>
<td>(null)</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>103-APR-11</td>
<td>STRT</td>
<td>19</td>
<td>19</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>104-APR-11</td>
<td>X</td>
<td>18</td>
<td>18</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>105-APR-11</td>
<td>X</td>
<td>17</td>
<td>17</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>106-APR-11</td>
<td>Y</td>
<td>20</td>
<td>20</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>107-APR-11</td>
<td>W</td>
<td>17</td>
<td>17</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>108-APR-11</td>
<td>Z</td>
<td>20</td>
<td>20</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>109-APR-11</td>
<td>AVG</td>
<td>16</td>
<td>16</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>(null)</td>
<td>10-APR-11 (null)</td>
<td>15</td>
<td>(null)</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>211-APR-11</td>
<td>STRT</td>
<td>15</td>
<td>15</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>212-APR-11</td>
<td>X</td>
<td>12</td>
<td>12</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>213-APR-11</td>
<td>X</td>
<td>11</td>
<td>11</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>214-APR-11</td>
<td>Y</td>
<td>15</td>
<td>15</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>215-APR-11</td>
<td>W</td>
<td>12</td>
<td>12</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>216-APR-11</td>
<td>Z</td>
<td>16</td>
<td>16</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>217-APR-11</td>
<td>AVG</td>
<td>14</td>
<td>14</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>(null)</td>
<td>18-APR-11 (null)</td>
<td>12</td>
<td>(null)</td>
<td>(null)</td>
<td></td>
</tr>
<tr>
<td>OSCORP</td>
<td>(null)</td>
<td>19-APR-11 (null)</td>
<td>11</td>
<td>(null)</td>
<td>(null)</td>
<td></td>
</tr>
</tbody>
</table>
Backtracking

• Non-deterministic state machine captures state at each row at pattern evaluation time and pushes details into stack
  – Backtracking simply walks back through the stack, looking for possible re-evaluation

• Moving forward we put more and more rows into the stack
  – Repeated within each partition

• Depending on complexity of pattern this can become memory-consuming
  – Chance to run out of PGA (ORA-30009) for large, complex pattern matching statements
  – Circumvent such situations by allocating more memory or...
  – Consider simplify your pattern!
NAMING BLOCKS
Keeping your code neatly and clearly defined for the next developer who has to work on it...
Name your blocks - not mandatory, *just helpful*

```
SELECT mr.symbol, mr.first_x, mr.last_z
FROM ticker t
MATCH_RECOGNIZE (
    PARTITION BY symbol ORDER BY tstamp
    MEASURES FIRST(x.tstamp) AS first_x,
        LAST(z.tstamp) AS last_z,
        tstamp AS tstamp
    ONE ROW PER MATCH
    PATTERN (X+ Y+ W+ Z+)
    DEFINE X AS (price < PREV(price)),
        Y AS (price > PREV(price)),
        W AS (price < PREV(price)),
        Z AS (price > PREV(price)) AND
        z.tstamp - FIRST(x.tstamp) <= 7
    )
WHERE mr.symbol IN ('ACME', 'OSCORP')
AND mr.tstamp BETWEEN '01-Apr-11' AND '09-Apr-11';
```

- Name your blocks
- Explicitly mark the MATCH_RECOGNIZE block
  - Reference in SELECT clause and predicates
- Makes life a lot *easier* for person trying to update your code
Don’t rely on DEFAULTS or exclude optional keywords!

• Many keywords/clauses have default values
  – *AFTER MATCH SKIP*…. default is *PAST LAST ROW*
  – *ALL ROWS PER MATCH*…. Default is *SHOW EMPTY MATCHES*

• Some keywords/clauses are optional
  – *PARTITION BY* and *ORDER BY*: don’t expect source data to be in correct order!
  – *MEASURES*: most patterns contain useful data points so add additional measures
    • *MATCH_NUMBER* and *CLASSIFIER* are useful as debugging tools but probably need to be removed
TESTING YOUR PATTERN

Build data sets that test all possibilities for matching your pattern to avoid unexpected results
You will need three data sets for testing...

1. Data set containing no matches so you can make sure that your pattern does not unexpectedly find patterns

2. Data set containing partial matches so you can make sure that your pattern does not consume all your PGA resources and generate an ORA-3009 error. Check your greedy/reluctant quantifiers are matching as expected

3. Data set contains at least one complete match (preferably a lot more) for your pattern
Summary

- SQL Pattern Matching is a powerful tool
- How to use most important keywords
- Key things to consider before you run a MATCH_RECOGINIZE query
- How state machines work and impact of backtracking
- Go and use SQL Pattern Matching to your advantage!
Try MATCH_RECOGNIZE today using livesql.us.oracle.com

Large selected of pattern matching scripts and tutorials on livesql.us.oracle.com