



DEEP DIVE INTO 12c MATCH_RECOGNIZE

Getting inside the SQL pattern
matching process

Hermann Bär

Director of Product Management
Data Warehousing

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Agenda

- 1 SQL Pattern Matching – quick recap
- 2 Using built-in measures to understand your pattern
- 3 Greedy vs. reluctant quantifiers
- 4 Understanding state machines
- 5 Backtracking
- 6 Summary



SQL Pattern Matching

Quick recap

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

3

Pattern matching in sequences of rows

The Challenge – a real-world business problem

“ ... detect if a phone card went from phone A to phone B to phone C... and back to phone A within ‘N’ hours... ”

“... and detect if pattern above occurs at least ‘N’ times within 7 days ...”

- Currently pattern recognition in SQL is difficult
 - Use multiple self joins (not good for *)
 - T1.handset_id <> T2.handset_id <>T3.handset_id AND.... T1.sim_id= ‘X’ AND T2.time BETWEEN T1.time and T1.time+2....
 - Use recursive query for * (WITH clause, CONNECT BY)
 - Use Window Functions (likely with multiple query blocks)

Sample Pattern Matching SQL

Finding V shaped patterns in a ticker stream

```
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

Sample Pattern Matching SQL

Finding V shaped patterns in a ticker stream

```
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

Sample Pattern Matching SQL

Finding V shaped patterns in a ticker stream

```
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

Variable definition

Sample Pattern Matching SQL

Finding V shaped patterns in a ticker stream

```
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

Measures

Variable names and operators

Variable definition

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!



Built-in Measures

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

Two key built-in measures

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

	SYMBOL	TSTAMP	PRICE	MN	PATTERN
PARTITION 1	1 ACME	05-APR-11	25	1	STRT
	2 ACME	06-APR-11	12	1	DOWN
	3 ACME	07-APR-11	15	1	JP
	4 ACME	08-APR-11	20	1	JP
	5 ACME	09-APR-11	24	1	JP
	6 ACME	10-APR-11	25	1	JP
	7 ACME	11-APR-11	19	2	STRT
	8 ACME	12-APR-11	15	2	DOWN
	9 ACME	13-APR-11	25	2	JP
	10 ACME	14-APR-11	25	2	JP
	11 ACME	15-APR-11	14	3	STRT
	12 ACME	16-APR-11	12	3	DOWN
	13 ACME	17-APR-11	14	3	JP
	14 ACME	18-APR-11	24	3	JP
PARTITION 2	15 GLOBEX	05-APR-11	25	1	STRT
	16 GLOBEX	06-APR-11	12	1	DOWN

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**
- Note: CLASSIFIER() cannot be used with **ONE ROW PER MATCH**
 - Not applicable to aggregated result

	SYMBOL	TSTAMP	PRICE	MN	PATTERN
1	ACME	05-APR-11	25	1	STRT
2	ACME	06-APR-11	12	1	DOWN
3	ACME	07-APR-11	15	1	UP
4	ACME	08-APR-11	20	1	UP
5	ACME	09-APR-11	24	1	UP
6	ACME	10-APR-11	25	1	UP
7	ACME	11-APR-11	19	2	STRT
8	ACME	12-APR-11	15	2	DOWN
9	ACME	13-APR-11	25	2	UP
10	ACME	14-APR-11	25	2	UP
11	ACME	15-APR-11	14	3	STRT
12	ACME	16-APR-11	12	3	DOWN
13	ACME	17-APR-11	14	3	UP
14	ACME	18-APR-11	24	3	UP
15	GLOBEX	05-APR-11	25	1	STRT
16	GLOBEX	06-APR-11	12	1	DOWN



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*

Wikipedia: https://en.wikipedia.org/wiki/Regular_expression

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 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

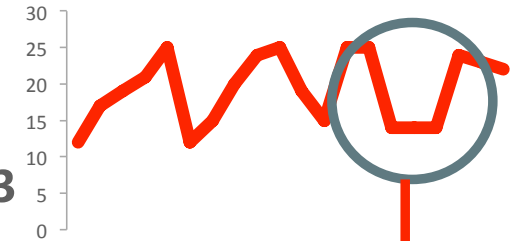
Finding V shaped patterns in a ticker stream using plus-sign greedy quantifier

```
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 the “+” greedy
quantifier

Using greedy quantifiers

Matching to variable DOWNS takes precedence over UP on row 13



	SYMBOL	TSTAMP	PRICE	MN	PATTERN	FIRST_DOWN	FIRST_PRICE	LAST_UP	LAST_PRICE
1	ACME	05-APR-11	25	1	STRT	05-APR-11	25	(null)	(null)
2	ACME	06-APR-11	12	1	DOWN	05-APR-11	25	(null)	(null)
3	ACME	07-APR-11	15	1	UP	05-APR-11	25	07-APR-11	15
4	ACME	08-APR-11	20	1	UP	05-APR-11	25	08-APR-11	20
5	ACME	09-APR-11	24	1	UP	05-APR-11	25	09-APR-11	24
6	ACME	10-APR-11	25	1	UP	05-APR-11	25	10-APR-11	25
7	ACME	11-APR-11	19	2	STRT	11-APR-11	19	(null)	(null)
8	ACME	12-APR-11	15	2	DOWN	11-APR-11	19	(null)	(null)
9	ACME	13-APR-11	25	2	UP	11-APR-11	19	13-APR-11	25
10	ACME	14-APR-11	25	2	UP	11-APR-11	19	14-APR-11	25
11	ACME	15-APR-11	14	3	STRT	15-APR-11	14	(null)	(null)
12	ACME	16-APR-11	14	3	DOWN	15-APR-11	14	(null)	(null)
13	ACME	17-APR-11	14	3	DOWN	15-APR-11	14	(null)	(null)
14	ACME	18-APR-11	24	3	UP	15-APR-11	14	18-APR-11	24

Conflict: 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

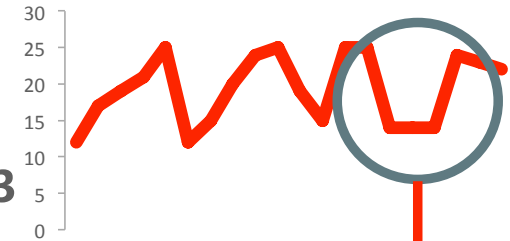
Finding V shaped patterns in a ticker stream using question-mark reluctant quantifier

```
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;
```

What happens if DOWN
uses ? to make quantifier
'reluctant' ...

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)

	SYMBOL	TSTAMP	PRICE	MN	PATTERN	FIRST_DOWN	FIRST_PRICE	LAST_UP	LAST_PRICE
1	ACME	05-APR-11	25	1	STRT	05-APR-11	25	(null)	(null)
2	ACME	06-APR-11	12	1	DOWN	05-APR-11	25	(null)	(null)
3	ACME	07-APR-11	15	1	UP	05-APR-11	25	07-APR-11	15
4	ACME	08-APR-11	20	1	UP	05-APR-11	25	08-APR-11	20
5	ACME	09-APR-11	24	1	UP	05-APR-11	25	09-APR-11	24
6	ACME	10-APR-11	25	1	UP	05-APR-11	25	10-APR-11	25
7	ACME	11-APR-11	19	2	STRT	11-APR-11	19	(null)	(null)
8	ACME	12-APR-11	15	2	DOWN	11-APR-11	19	(null)	(null)
9	ACME	13-APR-11	25	2	UP	11-APR-11	19	13-APR-11	25
10	ACME	14-APR-11	25	2	UP	11-APR-11	19	14-APR-11	25
11	ACME	15-APR-11	14	3	STRT	15-APR-11	14	(null)	(null)
12	ACME	16-APR-11	14	3	DOWN	15-APR-11	14	(null)	(null)
13	ACME	17-APR-11	14	3	UP	15-APR-11	14	17-APR-11	14
14	ACME	18-APR-11	24	3	UP	15-APR-11	14	18-APR-11	24



Understanding State Machines

...and why you need to care about them!

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

25

MATCH_RECOGNIZE and State Machines

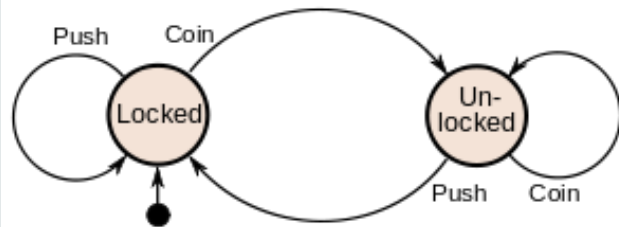
- Compilation phase generates a finite state machine FSM

*A finite-state machine (FSM) or finite-state automaton (plural: automata), or simply a **state machine**, is a mathematical **model of computation** used to design both computer programs and sequential logic circuits. 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**; the state it is in at any given time is called the current state. It can **change** 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.

Reference from wikipedia - https://en.wikipedia.org/wiki/Finite-state_machine

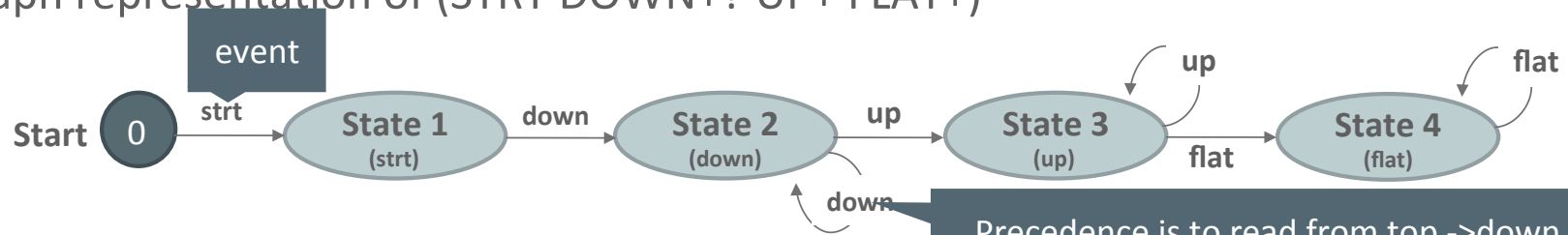
Turnstile State Machine



- 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

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



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

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 like memoization*

* <https://en.wikipedia.org/wiki/Memoization>

MATCH_RECOGNIZE plans based on State Machines

Explain plan indicates which algorithm is used:

Script Output x | Query Result x | Explain Plan x | Autotrace x
SQL | 0.202 seconds

OPERATION

- SELECT STATEMENT
 - SORT (ORDER BY)
 - VIEW
 - MATCH_RECOGNIZE (SORT)**
 - TABLE ACCESS (FULL)
 - Filter Predicates
 - SYMBOL='ACME'

Script Output x | Explain Plan x | Query Result x | Autotrace x
SQL | 0.575 seconds

OPERATION

- SELECT STATEMENT
 - SORT (ORDER BY)
 - VIEW
 - MATCH_RECOGNIZE (SORT DETERMINISTIC FINITE AUTO)**
 - TABLE ACCESS (FULL)
 - Filter Predicates
 - SYMBOL='ACME'

New keywords in explain plans

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

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

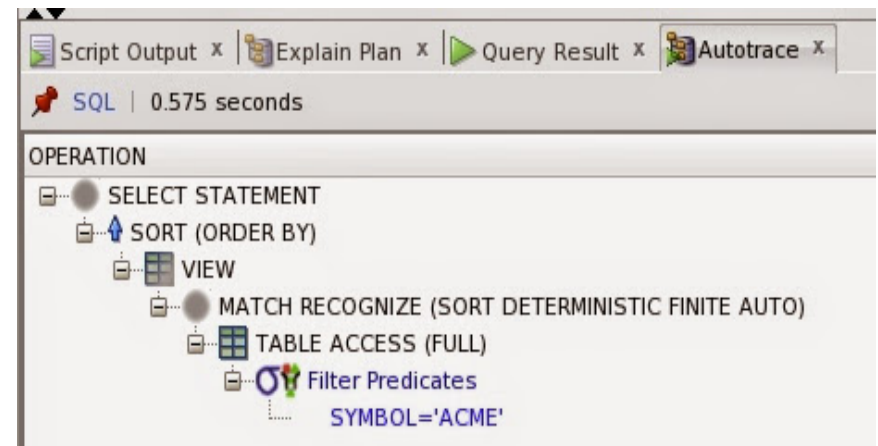
Optimizer keywords

- SORT – input data needs to be sorted before executing the state machine constructed for pattern recognition
- BUFFER – rows of input table expression come in the order required by MATCH_RECOGNIZE
 - Sort is used during pattern recognition just for buffering purpose

Deterministic state machine

No backtracking and runs in linear time

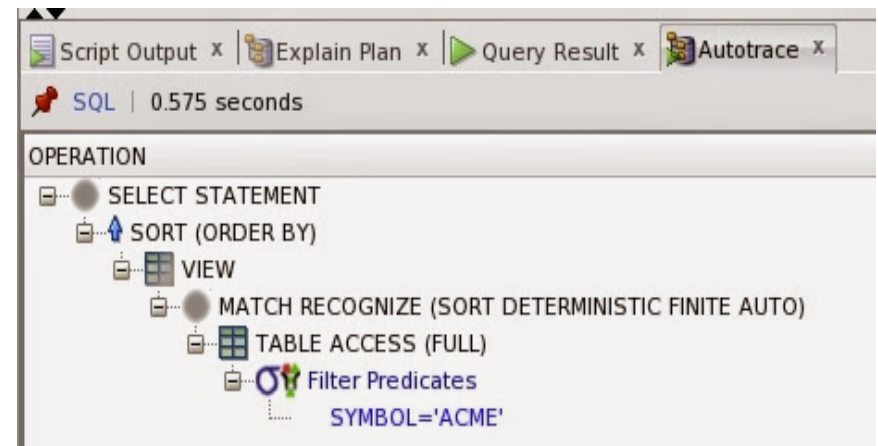
```
SELECT *
FROM Ticker
MATCH_RECOGNIZE (
  PARTITION BY symbol ORDER BY tstamp
  MEASURES strt.tstamp AS start_tstamp,
            LAST(down.tstamp) AS b_tstamp,
            LAST(up.tstamp) AS e_tstamp
  ONE ROW PER MATCH
  AFTER MATCH SKIP TO LAST y
  PATTERN (strt down up)
  DEFINE
    down AS price < PREV(price),
    up AS price > PREV(price)
) WHERE symbol= 'ACME' ;
```



Deterministic state machine

No backtracking and runs in linear time

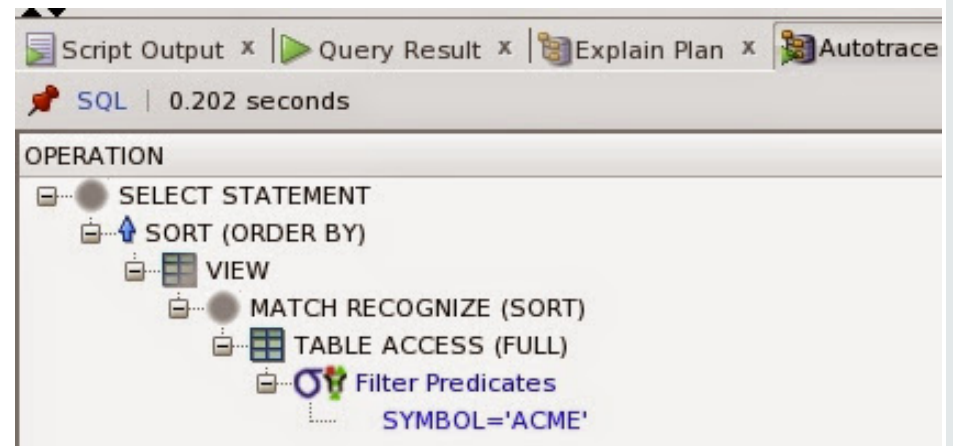
```
SELECT *
FROM Ticker
MATCH_RECOGNIZE (
  PARTITION BY symbol ORDER BY tstamp
  MEASURES strt.tstamp AS start_tstamp,
            LAST(down.tstamp) AS b_tstamp,
            LAST(up.tstamp) AS e_tstamp
  ONE ROW PER MATCH
  PATTERN (strt down up*)
  DEFINE
    down AS price < PREV(price),
    up   AS price > PREV(price)
) WHERE symbol= 'ACME' ;
```



Non-deterministic state machine

Determinism unknown, backtracking in place

```
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' ;
```



Backtracking

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

36

Setting the scene – searching for ‘W’ patterns

```
SELECT symbol, mn, tstamp, pattern, price,  
first_price, first_price, 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_x,  
             FIRST(strt.price) AS first_price,  
             LAST(z.tstamp) AS last_z,  
             LAST(z.price) AS last_price  
    ALL ROWS PER MATCH WITH UNMATCHED ROWS  
    PATTERN (strt 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))  
WHERE symbol = 'OSCORP';
```

Results for W-pattern search

	SYMBOL	MN	TSTAMP	PATTERN	PRICE	FIRST_PRICE	LAST_PRICE
1	OSCORP	(null)	01-APR-11	(null)	22	(null)	(null)
2	OSCORP	1	02-APR-11	STRT	22	22	(null)
3	OSCORP	1	03-APR-11	X	19	22	(null)
4	OSCORP	1	04-APR-11	X	18	22	(null)
5	OSCORP	1	05-APR-11	X	17	22	(null)
6	OSCORP	1	06-APR-11	Y	20	22	(null)
7	OSCORP	1	07-APR-11	W	17	22	(null)
8	OSCORP	1	08-APR-11	Z	20	22	20
9	OSCORP	(null)	09-APR-11	(null)	16	(null)	(null)
10	OSCORP	(null)	10-APR-11	(null)	15	(null)	(null)
11	OSCORP	2	11-APR-11	STRT	15	15	(null)
12	OSCORP	2	12-APR-11	X	12	15	(null)
13	OSCORP	2	13-APR-11	X	11	15	(null)
14	OSCORP	2	14-APR-11	Y	15	15	(null)
15	OSCORP	2	15-APR-11	W	12	15	(null)
16	OSCORP	2	16-APR-11	Z	16	15	16
17	OSCORP	(null)	17-APR-11	(null)	14	(null)	(null)
18	OSCORP	(null)	18-APR-11	(null)	12	(null)	(null)
19	OSCORP	(null)	19-APR-11	(null)	11	(null)	(null)

- Points to note:
- Line 1 is not matched because the 2nd element of our pattern, the first down test X, fails since price on line 1 is the same as price on line 2
- Pattern STRT is matched at line 2
 - Matching continues until line 8
 - Line 8 completes the first “W”
- Non-deterministic state machine
 - Backtracking will be used

Extending the pattern to test final price vs. initial price

```
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

What's going on ?

	SYMBOL	MN	TSTAMP	PATTERN	PRICE	FIRST_PRICE	LAST_PRICE
1	OSCORP	(null)	01-APR-11	(null)	22	(null)	(null)
2	OSCORP	1	02-APR-11	STRT	22	22	(null)
3	OSCORP	1	03-APR-11	X	19	22	(null)
4	OSCORP	1	04-APR-11	X	18	22	(null)
5	OSCORP	1	05-APR-11	X	17	22	(null)
6	OSCORP	1	06-APR-11	Y	20	22	(null)
7	OSCORP	1	07-APR-11	W	17	22	(null)
8	OSCORP	1	08-APR-11	Z	20	22	20
9	OSCORP	(null)	09-APR-11	(null)	16	(null)	(null)
10	OSCORP	(null)	10-APR-11	(null)	15	(null)	(null)
11	OSCORP	2	11-APR-11	STRT	15	15	(null)
12	OSCORP	2	12-APR-11	X	12	15	(null)
13	OSCORP	2	13-APR-11	X	11	15	(null)
14	OSCORP	2	14-APR-11	Y	15	15	(null)
15	OSCORP	2	15-APR-11	W	12	15	(null)
16	OSCORP	2	16-APR-11	Z	16	15	16
17	OSCORP	(null)	17-APR-11	(null)	14	(null)	(null)
18	OSCORP	(null)	18-APR-11	(null)	12	(null)	(null)
19	OSCORP	(null)	19-APR-11	(null)	11	(null)	(null)

	SYMBOL	MN	TSTAMP	PATTERN	PRICE	FIRST_PRICE	LAST_PRICE
1	OSCORP	(null)	01-APR-11	(null)	22	(null)	(null)
2	OSCORP	(null)	02-APR-11	(null)	22	(null)	(null)
3	OSCORP	1	03-APR-11	STRT	19	19	(null)
4	OSCORP	1	04-APR-11	X	18	19	(null)
5	OSCORP	1	05-APR-11	X	17	19	(null)
6	OSCORP	1	06-APR-11	Y	20	19	(null)
7	OSCORP	1	07-APR-11	W	17	19	(null)
8	OSCORP	1	08-APR-11	Z	20	19	20
9	OSCORP	1	09-APR-11	AVGP	16	19	20
10	OSCORP	(null)	10-APR-11	(null)	15	(null)	(null)
11	OSCORP	2	11-APR-11	STRT	15	15	(null)
12	OSCORP	2	12-APR-11	X	12	15	(null)
13	OSCORP	2	13-APR-11	X	11	15	(null)
14	OSCORP	2	14-APR-11	Y	15	15	(null)
15	OSCORP	2	15-APR-11	W	12	15	(null)
16	OSCORP	2	16-APR-11	Z	16	15	16
17	OSCORP	2	17-APR-11	AVGP	14	15	16
18	OSCORP	(null)	18-APR-11	(null)	12	(null)	(null)
19	OSCORP	(null)	19-APR-11	(null)	11	(null)	(null)

Comparing results of first and second statement

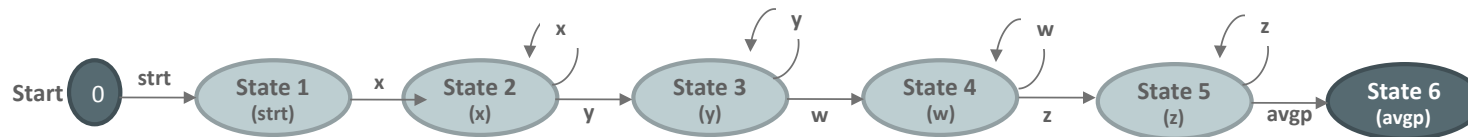
	SYMBOL	MN	TSTAMP	PATTERN	PRICE	FIRST_PRICE	LAST_PRICE
1	OSCORP	(null)	01-APR-11	(null)	22	(null)	(null)
2	OSCORP	(null)	02-APR-11	(null)	22	(null)	(null)
3	OSCORP	1	03-APR-11	STRT	19	19	(null)
4	OSCORP	1	04-APR-11	X	18	19	(null)
5	OSCORP	1	05-APR-11	X	17	19	(null)
6	OSCORP	1	06-APR-11	Y	20	19	(null)
7	OSCORP	1	07-APR-11	W	17	19	(null)
8	OSCORP	1	08-APR-11	Z	20	19	20
9	OSCORP	1	09-APR-11	AVGP	16	19	20
10	OSCORP	(null)	10-APR-11	(null)	15	(null)	(null)
11	OSCORP	2	11-APR-11	STRT	15	15	(null)
12	OSCORP	2	12-APR-11	X	12	15	(null)
13	OSCORP	2	13-APR-11	X	11	15	(null)
14	OSCORP	2	14-APR-11	Y	15	15	(null)
15	OSCORP	2	15-APR-11	W	12	15	(null)
16	OSCORP	2	16-APR-11	Z	16	15	16
17	OSCORP	2	17-APR-11	AVGP	14	15	16
18	OSCORP	(null)	18-APR-11	(null)	12	(null)	(null)
19	OSCORP	(null)	19-APR-11	(null)	11	(null)	(null)

Notable differences

- Row 2 is not matched anymore to always-true event STRT
- STRT variable now matched at row 3.
- W-pattern still ends at row 8
- Row 9 is now mapped to variable AVGP
- **Backtracking in action**

Different pattern match due to non-deterministic state

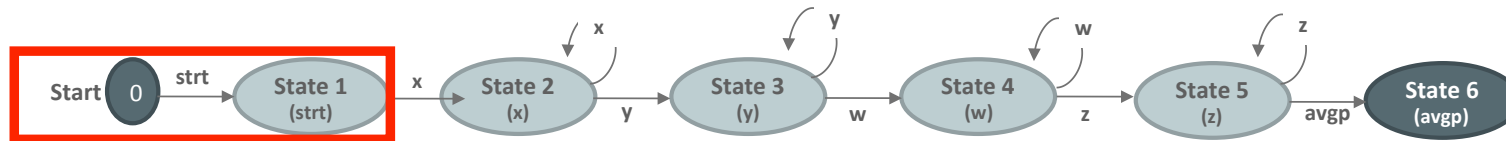
(STRT X+ Y+ W+ Z+ AVGP)



	SYMBOL	MN	TSTAMP	PATTERN	PRICE
1	OSCORP	(null)	01-APR-11	(null)	22
2	OSCORP	1	02-APR-11	STRT	22
3	OSCORP	1	03-APR-11	X	19
4	OSCORP	1	04-APR-11	X	18
5	OSCORP	1	05-APR-11	X	17
6	OSCORP	1	06-APR-11	Y	20
7	OSCORP	1	07-APR-11	W	17
8	OSCORP	1	08-APR-11	Z	20
9	OSCORP	(null)	09-APR-11	(null)	16

Different pattern match due to non-deterministic state

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



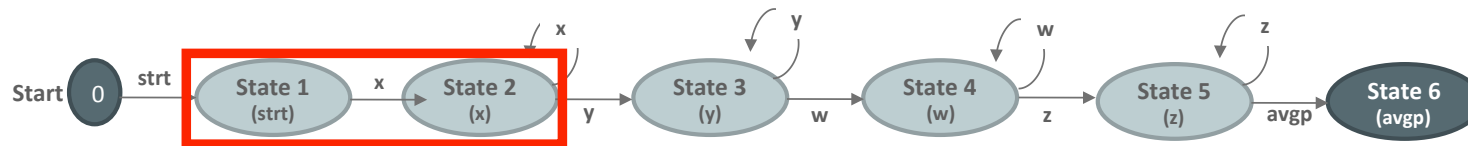
	SYMBOL	MN	TSTAMP	PATTERN	PRICE
1	OSCORP	(null)	01-APR-11	(null)	22
2	OSCORP	1	02-APR-11	STRT	22
3	OSCORP	1	03-APR-11	X	19
4	OSCORP	1	04-APR-11	X	18
5	OSCORP	1	05-APR-11	X	17
6	OSCORP	1	06-APR-11	Y	20
7	OSCORP	1	07-APR-11	W	17
8	OSCORP	1	08-APR-11	Z	20
9	OSCORP	(null)	09-APR-11	(null)	16

Current state	Row evaluated	Event	Event met	New state
0	2	strt	Y	1

- Start of pattern evaluation
 - Strt event true for row 2

Different pattern match due to non-deterministic state

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



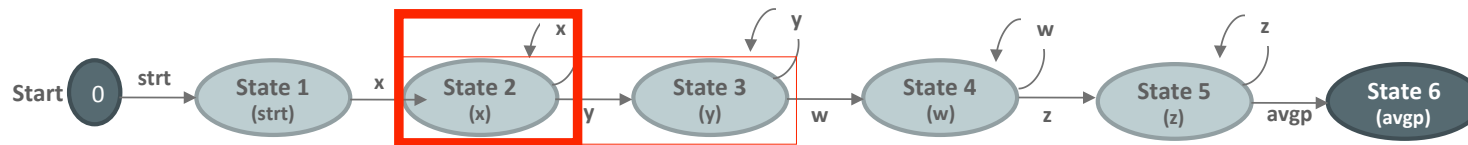
	SYMBOL	MN	TSTAMP	PATTERN	PRICE
1	OSCORP	(null)	01-APR-11	(null)	22
2	OSCORP	1	02-APR-11	STRT	22
3	OSCORP	1	03-APR-11	X	19
4	OSCORP	1	04-APR-11	X	18
5	OSCORP	1	05-APR-11	X	17
6	OSCORP	1	06-APR-11	Y	20
7	OSCORP	1	07-APR-11	W	17
8	OSCORP	1	08-APR-11	Z	20
9	OSCORP	(null)	09-APR-11	(null)	16

Current state	Row evaluated	Event	Event met	New state
0	2	strt	Y	1
1	3	X	Y	2

- Evaluation row 3
 - Event x true

Different pattern match due to non-deterministic state

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



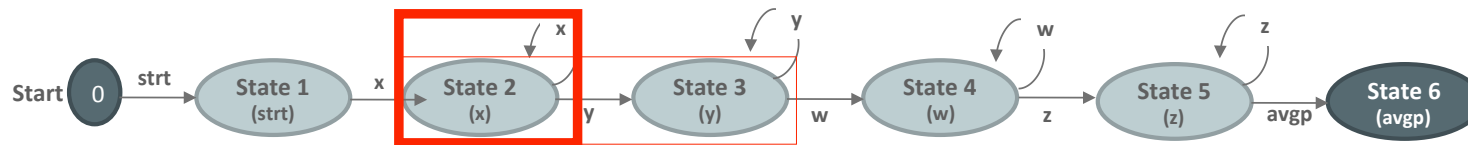
	SYMBOL	MN	TSTAMP	PATTERN	PRICE
1	OSCORP	(null)	01-APR-11	(null)	22
2	OSCORP	1	02-APR-11	STRT	22
3	OSCORP	1	03-APR-11	X	19
4	OSCORP	1	04-APR-11	X	18
5	OSCORP	1	05-APR-11	X	17
6	OSCORP	1	06-APR-11	Y	20
7	OSCORP	1	07-APR-11	W	17
8	OSCORP	1	08-APR-11	Z	20
9	OSCORP	(null)	09-APR-11	(null)	16

Current state	Row evaluated	Event	Event met	New state
0	2	strt	Y	1
1	3	X	Y	2
2	4	X	Y	2

- Evaluation row 4
 - Event X greedy, considered before Y, true

Different pattern match due to non-deterministic state

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



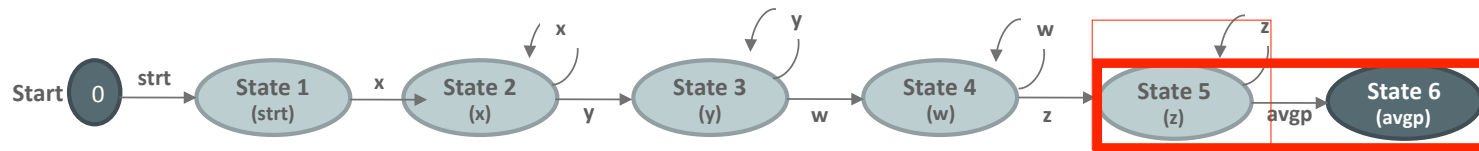
	SYMBOL	MN	TSTAMP	PATTERN	PRICE
1	OSCORP	(null)	01-APR-11	(null)	22
2	OSCORP	1	02-APR-11	STRT	22
3	OSCORP	1	03-APR-11	X	19
4	OSCORP	1	04-APR-11	X	18
5	OSCORP	1	05-APR-11	X	17
6	OSCORP	1	06-APR-11	Y	20
7	OSCORP	1	07-APR-11	W	17
8	OSCORP	1	08-APR-11	Z	20
9	OSCORP	(null)	09-APR-11	(null)	16

Current state	Row evaluated	Event	Event met	New state
0	2	strt	Y	1
1	3	X	Y	2
2	4	X	Y	2
2	5	X	Y	2

- Evaluation row 5
 - Event X greedy, considered before Y, true

Different pattern match due to non-deterministic state

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



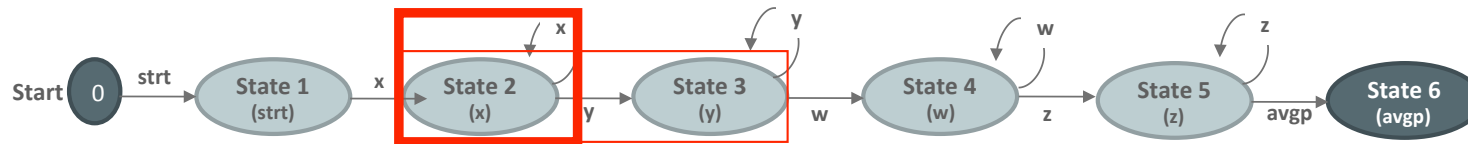
	SYMBOL	MN	TSTAMP	PATTERN	PRICE
1	OSCORP	(null)	01-APR-11	(null)	22
2	OSCORP	1	02-APR-11	STRT	22
3	OSCORP	1	03-APR-11	X	19
4	OSCORP	1	04-APR-11	X	18
5	OSCORP	1	05-APR-11	X	17
6	OSCORP	1	06-APR-11	Y	20
7	OSCORP	1	07-APR-11	W	17
8	OSCORP	1	08-APR-11	Z	20
9	OSCORP	(null)	09-APR-11	(null)	16

Current state	Row evaluated	Event	Event met	New state
0	2	strt	Y	1
1	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
5	9	Avgp (2 - 8)	N	FAIL

- Event Z false for row 9, evaluation of event AVGP for rows 2 – 8
 - Event AVGP is false, pattern match FAIL
 - Backtracking takes place

Different pattern match due to non-deterministic state

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



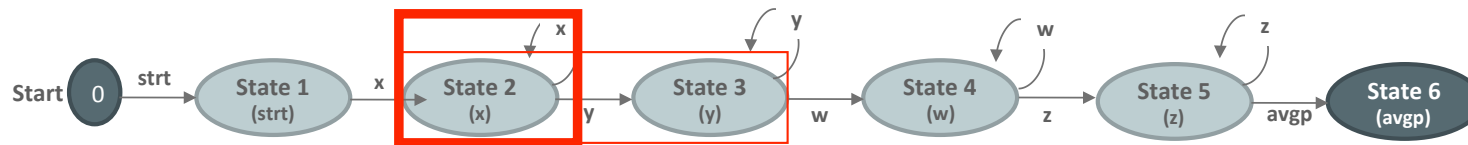
	SYMBOL	MN	TSTAMP	PATTERN	PRICE
1	OSCORP	(null)	01-APR-11	(null)	22
2	OSCORP	1	02-APR-11	STRT	22
3	OSCORP	1	03-APR-11	X	19
4	OSCORP	1	04-APR-11	X	18
5	OSCORP	1	05-APR-11	X	17
6	OSCORP	1	06-APR-11	Y	20
7	OSCORP	1	07-APR-11	W	17
8	OSCORP	1	08-APR-11	Z	20
9	OSCORP	(null)	09-APR-11	(null)	16

Current state	Row evaluated	Event	Event met	New state
0	2	str	Y	1
1	3	X	Y	2
2	4	X	Y	2
2	5	X	IGNORED	2
2	5	Y	N	FAIL

- Backtracking: find first row with possible alternative event
 - Row 5 re-evaluated with event Y, false
 - Pattern FAIL, more backtracking

Different pattern match due to non-deterministic state

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



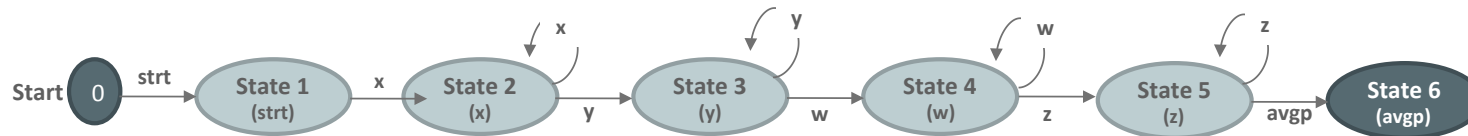
	SYMBOL	MN	TSTAMP	PATTERN	PRICE
1	OSCORP	(null)	01-APR-11	(null)	22
2	OSCORP	1	02-APR-11	STRT	22
3	OSCORP	1	03-APR-11	X	19
4	OSCORP	1	04-APR-11	X	18
5	OSCORP	1	05-APR-11	X	17
6	OSCORP	1	06-APR-11	Y	20
7	OSCORP	1	07-APR-11	W	17
8	OSCORP	1	08-APR-11	Z	20
9	OSCORP	(null)	09-APR-11	(null)	16

Current state	Row evaluated	Event	Event met	New state
0	2	strt	Y	1
1	3	X	Y	2
2	4	X	IGNORED	2
2	4	Y	N	FAIL

- More backtracking
 - Row 4 re-evaluated with event Y, false
 - Pattern FAIL, more backtracking

Different pattern match due to non-deterministic state

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



	SYMBOL	MN	TSTAMP	PATTERN	PRICE
1	OSCORP	(null)	01-APR-11	(null)	22
2	OSCORP	1	02-APR-11	STRT	22
3	OSCORP	1	03-APR-11	X	19
4	OSCORP	1	04-APR-11	X	18
5	OSCORP	1	05-APR-11	X	17
6	OSCORP	1	06-APR-11	Y	20
7	OSCORP	1	07-APR-11	W	17
8	OSCORP	1	08-APR-11	Z	20
9	OSCORP	(null)	09-APR-11	(null)	16

Current state	Row evaluated	Event	Event met	New state
0	2	Row skipped		
0	3	Strt	Y	1

- Backtracking fails
 - No more alternate events in row set 2 – 8
 - Pattern fail, row 2 skipped

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 for large, complex pattern matching statements
 - Circumvent such situations by allocating more memory

Summary



SQL Pattern Matching is a powerful tool



Easy data analysis out-of-the-box with built-in measures
MATCH_NUMBER and CLASSIFIER



The importance to understand greedy and reluctant
quantifiers



How do state machines and back tracking work



Go and use SQL Pattern Matching to your advantage!

ORACLE®